VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Detekce a odhad 3D pózy objektů v RGB-D obrazech

3D Object Detection and Pose Estimation in RGB-D Images

2018 Bc. Jan Šimeček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jan Šimeček**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Detekce a odhad 3D pózy objektů v RGB-D obrazech
3D Object Detection and Pose Estimation in RGB-D Images

Jazyk vypracování: čeština

Zásady pro vypracování:

Sledování objektů je důležitou součástí počítačového vidění. Cílem práce je implementace algoritmu pro rychlou detekci 3D objektů bez textur v obraze. Hledaný objekt je definován jako sada obrazů, které vyobrazují zadaný objekt pod různými úhly pohledu (pózami objektu). Ve výsledném obraze je pak objekt označen a s přiřazenou pravděpodobností k nalezenému vzoru.

Ve své práci proveďte:
1. Nastudujte aktuální stav poznání ohledně detekce objektů bez textur.
2. Naimplementujte algoritmus [1] pro detekci objektů ve scéně. Výstupem bude též určení pózy hledaných objektů.
3. Svou implementaci řádně otestujte a zdokumentujte.
4. Proveďte závěrečné shrnutí.

Seznam doporučené odborné literatury:

[1] T. Hodaň and X. Zabulis and M. Lourakis and Š Obdržálek and J. Matas: Detection and fine 3D pose estimation of texture-less objects in RGB-D images, Intelligent Robots and Systems (IROS), pp. 4421-4428, doi={10.1109/IROS.2015.7354005}, 2015

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Gaura, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 .....

Rád bych tímto poděkoval Ing. Janu Gaurovi, Ph.D. za cenné rady a přípomínky při tvorbě této práce.

Abstrakt

Práce se zabývá implementací algoritmu pro rychlou detekci 3D objektů bez textur a následným určením jejich pózy v obraze. Jednotlivé objekty jsou definovány sadou obrázků zachycujících je pod různými úhly (pózami). Práce je rozdělena na teoretickou a praktickou část. V teoretické části jsou představeny použité metody, dataset a související práce. Praktická část se věnuje samotné implementaci algoritmu a důkladně popisuje jednotlivé části detekční kaskády. Dále je zde uveden postup precizního určení pózy detekovaných objektů. Na závěr je provedeno závěrečné shrnutí zahrnující experimenty, evaluace a představení naměřených výsledků.

Klíčová slova: template matching, zpracování obrazu, detekce objektů, precizní odhad pózy, detekce objektivity, RGB-D obrazy

Abstract

The thesis deals with the implementation of the algorithm for fast detection of 3D textureless objects and subsequent determination of their pose in the image. Each object is represented by the set of images depicting the object from various angles (poses). The thesis is divided into theoretical and practical part. Theoretical part introduces used methods, dataset and related work. The practical part deals with the implementation of the algorithm itself and thoroughly describes the individual parts of the detection cascade. Procedure for fine pose estimation of detected objects is further described. Finally, a final summary is made, including experiments, evaluation and presentation of measured results.

Key Words: template matching, image processing, object detection, fine pose estimation, objectness detection, RGB-D images

Obsah

Seznam použitých zkratek a symbolů

SAD	–	Sum of Absolute Difference
SSD	–	Sum of Squared Difference
CC	–	Cross-Correlation
NCC	–	Normalized Cross-Correlation
NMS	–	Non-Maxima Suppression
MS	–	Microsoft
OpenCV	–	Open Source Computer Vision Library
OpenGL	–	Open Graphics Library
ICP	–	Iterative Closest Point
SSE	–	Streaming SIMD Extensions
CAD	–	Computer Aided Design
API	–	Application Programming Interface
ICP	–	Iterative Closest Point
PSO	–	Particle Swarm Optimization
GPU	–	Graphics Processing Unit
CPU	–	Central Processing Unit

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Detekce a sledování objektů tvoří důležitou součást počítačového vidění. Především objekty bez textury, ač všudypřítomné, a to zejména v průmyslu a robotických aplikacích, představují mnohdy výzvu pro moderní detekční algoritmy [1].

Častým použitím těchto algoritmů je identifikace a přesná lokalizace hledaného objektu, která je následně využita pro další manipulaci. Jednotně zabarvený a relativně neměnný povrch těchto objektů nese jen velmi malou, ne-li žádnou informaci, což značně ztěžuje jejich detekci.

Většina moderních detekčních algoritmů je schopna velice rychle identifikovat pozice možných kandidátů objektů ve scéně pomocí detekce takzvaných bodů zájmu[1][2]. Tento přístup nicméně mnohdy selže v případě objektů bez textury, jejichž vzhled je dán především jejich tvarem, vlastnostmi materiálu a konfigurací osvětlení. Tyto vlastnosti je však nutné znát předem, a je jednodušší reprezentovat jednotlivé objekty souborem několika obrázků, které jej zachycují ze všech možných úhlů. Výsledná detekce tedy buď najde objekt odpovídající jednomu z předem zachycených templatů, či nenajde žádného odpovídajícího kandidáta a okno lze klasifikovat jako obsahující pozadí.

V následujících několika sekcích bude přiblížen návrh řešením metody detekce více objektů bez textury a precizního odhadu jejich pózy, jehož algoritmus vychází zejména z metod popsaných v [3]. Tato metoda se snaží zajistit velkou spolehlivost s malým počtem špatných detekcí při zachování značné rychlosti. Vstupem je soubor RGB-D obrázků nasnímaných senzorem, jako je např. MS Kinect. Tyto senzory poskytují zarovnané hloubkové a barevné obrazy popisující jak vzhled, tak geometrii snímané scény.

1.1 Cíl práce

Cílem této diplomové práce je podrobně nastudovat oblast detekce a sledování objektů pomocí počítačového vidění. Konkrétně se jedná o implementaci detekce a precizního odhadu 3D pózy (polohy a rotace) předem známých objektů bez textury, s definovanou geometrií, v obrazech nasnímaných RGB-D senzorem (MS Kinect v2).

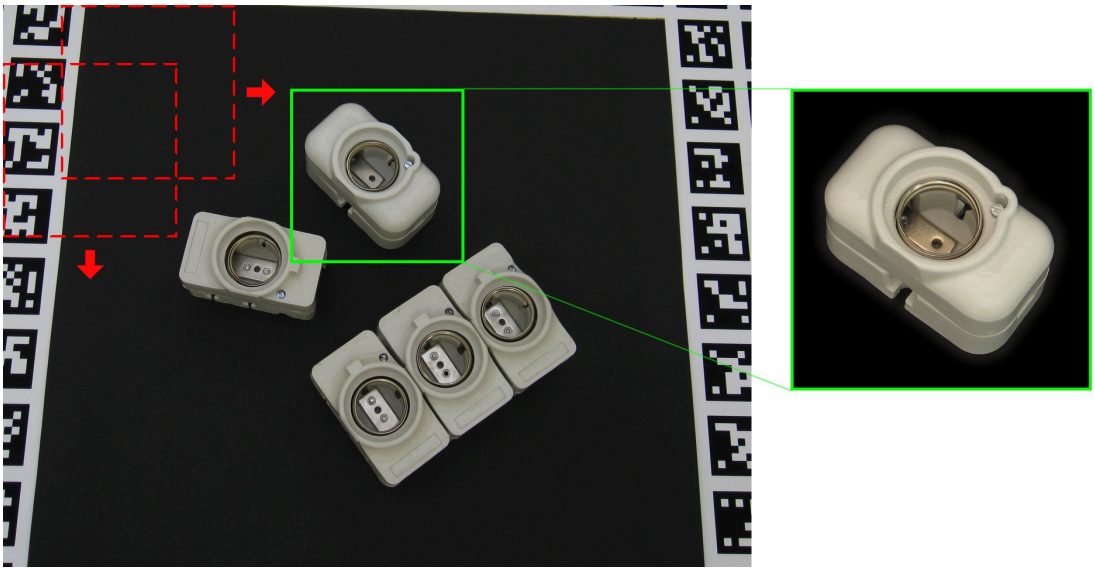
Detekční algoritmus provede nalezení objektů a vykreslení odpovídajících 3D modelů do sledovaného obrazu s precizním odhadem jejich orientace v 3D prostoru. Implementace algoritmu bude psána v jazyce C++ s využitím knihoven pro analýzu obrazu, jako je OpenCV, a práci s 3D grafikou s využitím OpenGL či rozšíření OpenCV v podobě VIZ. Takto implementovaný algoritmus najde využití především v oblasti průmyslu či robotiky pro provádění robotických montážních a manipulačních úloh.

[1] Bod zájmu - představuje bod v obraze, který má jasně danou definici a je stabilní za různých podmínek osvětlení. Okolí bodu je zároveň bohaté na informace, které představuje většinou nějaká forma výrazné 2D textury.

2 Současné techniky template matchingu

Template matching je technika ve zpracování obrazu, pomocí níž lze vyhledávat dílčí obrázky ze vstupního obrazu, které odpovídají cílovému templatu, tedy šabloně či vzoru. Templaty bývají často používány pro tisk znaků [4], identifikaci čísel nebo jednoduchých objektů. Tato technika je široce používaná v oblastech detekce objektů, jako např. v robotice, kde může sloužit jako prostředek pro řízení robota, v lékařském zobrazování nebo ve výrobě, v podobě kontroly kvality výrobků.

Rozhodujícím bodem při použití template matchingu je definovat vhodné měření pro vyčíslení míry podobnosti mezi templatem a dílčím obrázkem vstupního obrazu. Tento proces však vyžaduje značnou výpočetní kapacitu, jelikož průběh porovnávání zahrnuje postupné posouvání posuvného okénka (sliding window) napříč celým obrazem a následným výpočtem podobnosti mezi templaty a obsahem posuvného okénka. Mezi možná urychlení patří zmenšení velikosti vstupního obrazu a templatu nebo posouvání posuvného okénka po předem definovaných skocích. Zároveň se často využívá obrazové pyramidy, která umožňuje detekovat hledaný objekt napříč několika měřítky.



Obrázek 1: Příklad template matchingu s využitím posuvného okénka.

Vzhledem k jejich špatné generalizaci a limitované aplikaci byly techniky založené na template matchingu nějaký čas vytlačeny metodami zabývajícími se vyhledáváním lokálních bodů zájmu, získaných z objektů bohatých na texturu [3]. Tyto přístupy vyžadují pouze malý počet templatů na objekt a zobecňují se velice dobře pro širokou škálu odlišných scén. S příchodem moderních počítačů vybavených většími pamětmi dnes metody založené na template matchingu opět získávají na popularitě. Je tak zcela běžné uchovávat až tisíce templatů na objekt, což umožňuje důkladně zachytit všechny vlastnosti daného objektu.

2.1 Kategorizace template matching metod

Obecná kategorizace se skládá z metod využívajících příznaků a metod založených na porovnávání templatů či oblastí. Další problémy, se kterými se lze při template matchingu setkat, jsou sledování pohybu a řešení okluze.

2.1.1 Metody využívající příznaků

Pokud má template silné příznaky (angl. features), kterými mohou být např. hrany, textury, rohy nebo body, může být vhodné upřednostnit jejich použití pro detekci objektů namísto jiných metod. V tomto případě při porovnávání není využito všech pixelů daného templatu, ale pouze vybraných bodů zájmu či oblastí. Výpočet se tak stává mnohem efektivnější při práci se vstupními obrazy vyšších rozlišení. Cílem je najít korespondující dvojice bodů příznaků mezi vstupním obrazem a templatem [5]. Zároveň lze tuto metodu použít i pokud je hledaný objekt ve vstupním obraze nějakým způsobem transformovaný (natočený).

2.1.2 Metody založené na oblastech

Metody založené na oblastech, často také nazývané korelační nebo template matching metody, spojují v jednom kroku detekci i porovnvání. Tyto techniky spravují obrázky bez snahy identifikovat v nich nějaký význačný předmět. Jsou často používány v případech, kdy templaty nemají žádné silné příznaky nebo velikost vstupního obrázku odpovídá velikosti samotného templatu. Pro odhad korespondence (míry podobnosti) se většinou používají okna s předem definovanou velikostí.

Metody založené na oblastech můžou potenciálně vyžadovat zpracování velkého množství pixelů. To lze omezit zmenšením vstupního obrazu i samotného templatu o stejný faktor a následně provést operaci porovnávání na zmenšených obrázcích (obrazová pyramida).

2.1.3 Sledování pohybu a řešení okluze

Pro templaty, které nemohou a nemusí poskytovat přímou shodu, je možné využít tzv. vlastního podprostoru (angl. eigenspace). Ten poskytuje templaty zachycující hledaný objekt pod různými podmínkami jako osvětlení, jiný barevný kontrast, měnící se perspektiva nebo další přijatelné vhodné pózy objektu. Například pokud by uživatel ve vstupním obraze vyhledával obličej, vlastní podprostor může obsahovat templaty, které jej zachycují pod různými úhly ke kameře v odlišných světelných podmínkách.

Dále může nastat situace, kdy je hledaný objekt v obraze zakryt jiným objektem nebo vznikají jiné problémy způsobené pohybem. V těchto případech není žádoucí poskytovat templaty pro každou situaci, která může nastat. Příkladem může být vyhledávání hracích karet, kde je hledaná karta zakryta prsty uživatele nebo jinou kartou. Vhodným řešením je rozdělit hledaný template do několika podobrázků a provést porovnání na jednotlivých částech zvlášť.

2.2 Měření míry podobnosti

Jak již bylo zmíněno v předchozí sekci 2.1.2, u template matchingu je očekáván na vstupu obraz a template představující hledaný objekt. Ten se postupně přikládá na všechny pozice v obraze, kde se počítá míra shody mezi templatem a pozicí okna v obraze. Tento koeficient lze vypočítat několika způsoby. Mezi nejčastější metody patří: suma absolutních rozdílů, suma čtvercových rozdílů, vzájemná korelace a její normalizovaná varianta [6].

2.2.1 Suma absolutních rozdílů

Sumu absolutních rozdílů (angl. Sum of Absolute Difference (SAD)) lze vypočítat získáním absolutních rozdílů napříč všemi pixely mezi vstupním obrazem S a odpovídající pozicí pixelu v templatu T s využitím rovnice (1). Sumu těchto hodnot lze následně použít jako koeficient míry podobnosti mezi obrázkem S a templatem T, kdy čím menší tato hodnota je, tím více se jednotlivé pixely na daných pozicích shodují. Tudíž lze předpokládat, že se zde nachází hledaný objekt.

$$\mathrm{SAD}(u,v) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} \left|T(m,n) - S(u+m, v\ +n)\right|. \tag{1}$$

Ve srovnání s ostatními metodami (SSD, NCC) je SAD přímočará, jednoduchá a výpočetně nenáročná. Může být však nespolehlivá a produkovat chybné výsledky v případě změn ve světelných podmínkách, barvě, velikosti či tvaru. Díky své rychlosti je však možné ji použít spolu s jinými metodami, jako je detekce hran, pro zlepšení spolehlivosti.

2.2.2 Suma čtvercových rozdílů

Suma čtvercových rozdílů (ang. Sum of Squared Difference (SSD)) představuje jednu z více používaných metod pro výpočet koeficientu míry podobnosti, lze ji vypočítat dle rovnice (2). Nejmenší hodnota pixelu opět představuje nejlepší shodu, jako tomu bylo v případě SAD.

$$\mathrm{SSD}(u,v) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} \left(T(m,n) - S(u+m, v\ +n)\right)^2. \tag{2}$$

Ve srovnání s SAD se jedná o výpočetně náročnější, z důvodů nutnosti násobení, ale stále velice používanou metodu. Převážně vzhledem ke své jednoduchosti a stále relativně malé výpočetní náročnosti. NCC však ve většině případů produkuje přesnější a spolehlivější výsledky.

2.2.3 Vzájemná korelace

Vzájemná korelace (angl. Cross-Correlation (CC)) představuje sumu párových násobků hodnot jednotlivých pixelů a lze ji vypočítat s využitím rovnice (3).

$$\mathrm{CC}(u,v) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(T(m,n)S(u+m,v\ +n))\,. \tag{3}$$

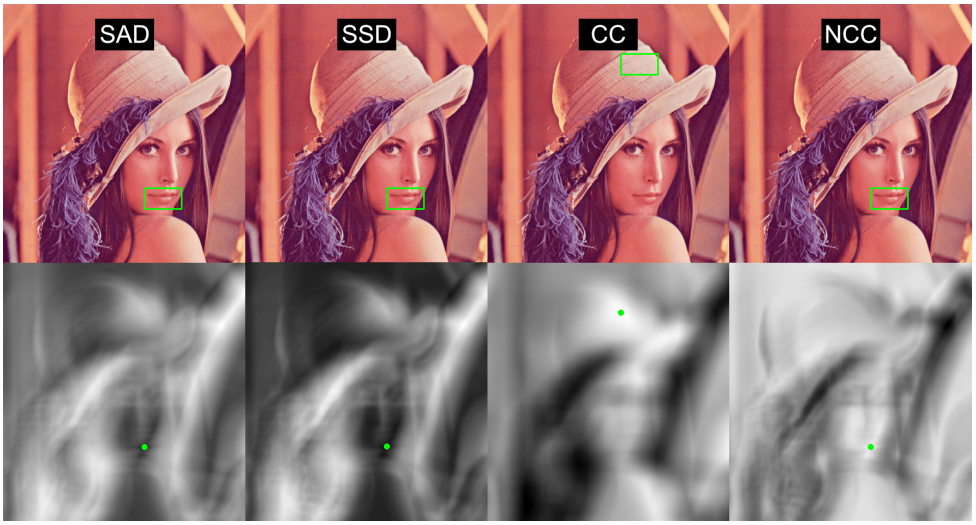
V reálných aplikacích se však tato metoda většinou nepoužívá. Přestože je relativně výpočetně nenáročná, tak vzhledem k její nespolehlivosti v případech, kdy se v obrázku nachází větší změny v měřítku nebo rotaci mezi hledaným objektem a vstupním obrazem, se využívá spíše její normalizovaná varianta, a to i přes daleko větší výpočetní náročnost.

2.2.4 Normalizovaná vzájemná korelace

Normalizovaná vzájemná korelace (angl. Normalized Cross-Correlation (NCC)) představuje jednu z nejpoužívanějších metod pro výpočet míry podobnosti mezi dvěma obrazy. Její největší výhodou oproti typické CC je větší odolnost vůči změnám v osvětlení scény, lze ji vypočítat s využitím následující rovnice:

$$\mathrm{NCC}(u,v) = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(T(m,n)S(u+m,v\ +n))}{\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}T(m,n)^2\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}S(u+m,v\ +n)^2}}\,.$$

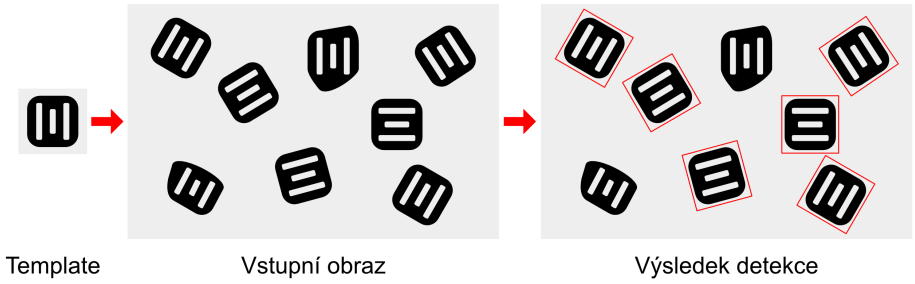
V porovnání s metodami SAD, SSD a CC je NCC ve většině situací nejpřesnější, avšak výpočetně mnohem náročnější. Při jejím použití je, pro urychlení výpočtu koeficientů, doporučeno provést jednoduché předfiltrování zpracovávaných oken např. aplikací kontroly salience [7].



Obrázek 2: Porovnání výsledků jednotlivých metod výpočtu míry podobnosti.

2.3 Pokročilé techniky

Všechny výše zmíněné metody bohužel trpí stejnými nedostatky. Při vyhledávání jsou výskyty vzorků ve vstupním obraze nuceny zachovat orientaci referenčního obrázku. Zároveň je velice neefektivní a časově náročné počítat korelaci mezi templatem a vstupním obrazem pro obrazy středních a vyšších rozlišení [4]. Existují proto techniky, které se snaží tyto nedostatky eliminovat.



Obrázek 3: Obrazová pyramida doplněná o různé rotace templatu.

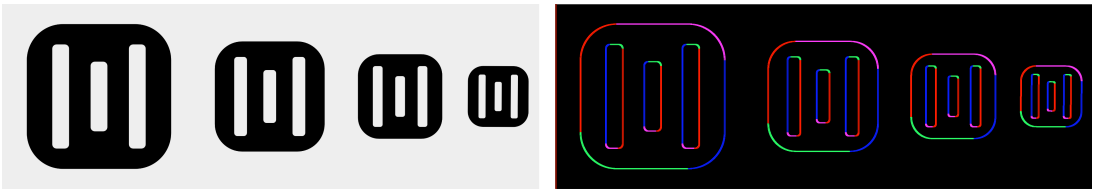
2.3.1 Rozšíření obrazových pyramid o orientaci templatů

Jedná se o rozšíření korelačních technik s cílem zlepšit jejich efektivitu o možnost vyhledávat objekty nezávisle na jejich orientaci. Přestože ve většině situací je hledaný objekt uniformní a fixní (nemění se jeho póza), často se stává, že detekované objekty jsou ve vstupním obraze natočeny. Při porovnávání je tak klasická obrazová pyramida rozšířena o možnost vyhledávání natočených instancí hledaného templatu.

Toho lze dosáhnout generováním ne jedné, ale hned několika obrazových pyramid. Jedné pro každou možnou rotaci templatu. Během vyhledávání poté algoritmus najde odpovídající pár (pozice templatu, orientace templatu) namísto pouze jeho pozice.

2.3.2 Template matching s využitím hran

Porovnávání s využitím hran vylepšuje výše zmíněnou metodu se znalostí jednoho zásadního pozorování - tvar jakéhokoliv objektu je definován z převážné většiny jeho hranami. Proto namísto porovnávání celého templatu, jsou nejprve z obrázku vyextrahovány hrany a porovnávány pouze blízké pixely. Je tak možné se vyhnout nadbytečným výpočtům. Ve většině reálných aplikací toto řešení přináší velký nárůst rychlosti ve srovnání s předchozími metodami.



Obrázek 4: Ukázka template matchingu s využitím hran.

3 Detekce hran

V předchozí sekci 2.3.2 již bylo zmíněno použití hran pro detekci objektů. V následujících několika sekcích budou popsány současné techniky jejich extrakce ze vstupních obrazů.

Detekce hran zahrnuje velké množství matematických metod, které se snaží v obraze identifikovat body, v nichž je náhlá změna jasu. Jinak řečeno, v těchto bodech vzniká nespojitost [8]. Body, v nichž dochází ke změnám jasu, jsou typicky organizovány do křivek, které představují hrany objektů ve scéně. Jedná se o podstatnou část zpracování obrazu a strojového vidění, a to zejména v oblastech detekce a extrakce vlastností objektů.

3.1 Vlastnosti hran

Cílem detekce náhlých změn jasu je zachytit důležité události ve změnách vlastností obrazu. Tyto změny typicky představují:

- nespojitost v hloubce nebo orientaci povrchu,
- změny ve vlastnostech materiálu,
- změny v osvětlení snímané scény.

Výsledkem detekce hran v obraze je v ideálním případě množina vzájemně propojených křivek, které reprezentují siluety jednotlivých objektů, nebo také nespojitosti v orientacích povrchu. Aplikace algoritmu pro detekci hran může tudíž značně redukovat méně důležité informace v obraze a přitom zachovat jeho strukturální vlastnosti.

Avšak detekce hran není v reálných situacích často ideální, jelikož její výsledek bývá ovlivněn šumem, který způsobuje značnou fragmentaci. Výsledkem toho je množina hran, které nejsou vzájemně propojené a mohou obsahovat i chybné hrany, které nejsou součástí oblasti zájmu. Proti šumu lze částečně bojovat s pomocí Gaussova nebo mediánového filtru, který obraz před samotnou detekcí mírně rozostří. Hrany získané z 2D obrazu snímajícího 3D scénu lze klasifikovat do dvou kategorií:

- **Nezávislé na aktuálním zobrazení** – tyto hrany typicky popisují vnitřní vlastnosti snímaného objektu, jako jsou: značení na jeho povrchu, vlastnosti materiálu nebo jeho tvar.
- **Závislé na aktuálním zobrazení** – tyto hrany se mohou měnit se změnami zobrazení. Popisují zejména obecnou geometrii scény a siluety jednotlivých objektů, které se vzájemně překrývají.

3.2 Metody první derivace

Většina detektorů hran je založena na měření intenzity gradientu [9] v bodě vstupního obrazu nebo jejich vyfiltrovaných verzích (pomocí jednoho z výše uvedených filtrů). Nejjednodušší přístup detekce gradientů je využití první derivace, která je v diskrétních obrazech nahrazena centrální diferencí:

$$d_x = \frac{I(x-1,y) - I(x+1,y)}{2}\,, \quad d_y = \frac{I(x,y-1) - I(x,y+1)}{2}\,. \tag{4}$$

Výsledek rovnice (4) udává směr největší změny gradientu, který směřuje směrem napříč hranou. Samotná hrana je kolmá k danému gradientu. Jako u kteréhokoliv jiného vektoru, i zde je možné zjistit jeho velikost (množství změn mezi pixely v jeho okolí – síla hrany) pomocí rovnice (5).

$$|d| = \sqrt{d_x^2 + d_y^2}\,. \tag{5}$$

Mnoho algoritmů využívá pouze velikosti gradientu, je ale důležité zmínit, že i jeho orientace může nést stejné množství informací. Vzhledem k tomu, že deriváty jsou lineární a invariantní vůči posuvu, výpočet gradientů bývá často nahrazen konvolucí. Existuje mnoho konvolučních masek [10], mezi které patří např. Sobelova, Robertsova, Kisrchova nebo maska Prewittové.

3.2.1 Robertsova maska

Jelikož je snaha v obraze najít rozdíl mezi dvěma sousedními pixely, jedním ze způsobů je explicitně použít operátory $\{1, -1\}$, které vypočítají rozdíl dvou sousedních pixelů. Robertsova maska, daná rovnicí (6), nedefinuje přímo deriváty vzhledem k ose x a y, ale vzhledem k dvěma diagonálním směrům.

$$K_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad K_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}. \tag{6}$$

Výslednou velikost gradientu je následně možné vypočítat jako délku vektoru pomocí rovnice (7). V praxi je však tato maska moc malá na to, aby dokázala spolehlivě detekovat hrany v obrazech postižených šumem.

$$|g| = \sqrt{(K_1 * f)^2 + (K_2 * f)^2}\,. \tag{7}$$

3.2.2 Kirschova (kompasová) maska

Kirschova (kompasová) maska, představuje nelineární detektor hran, který hledá gradienty v několika předdefinovaných směrech. Výchozí maska $K^{(1)}$ je tedy postupně otáčena o 45° napříč

všemi 8 směry kompasu (N, NE, E, SE, S, SW, W, NW):

$$K^{(1)} = \begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}, \quad K^{(2)} = \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix}, \quad K^{(3)} = \begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix}, \quad \cdots$$

Výsledná velikost gradientu je následně rovna maximální velikosti napříč všemi směry, lze ji vypočítat s využitím následující rovnice:

$$|g| = \max_{n=1,\ldots,8} (K^{(n)} * f)\,. \tag{8}$$

3.2.3 Maska Prewittové

Maska Prewittové, daná rovnicí (9), je založena na principu centrální diference. Provádí konvoluci ve směru x a y s využitím větší velikosti masky (3×3) a průměrování, které napomáhá k redukci šumu. Přesto však produkuje aproximace gradientů, které jsou relativně hrubé, zejména v případě vysokofrekvenčních variací v obraze.

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad K_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}. \tag{9}$$

3.2.4 Sobelova maska

Sobelova maska, definovaná rovnicí (10), se znovu spoléhá na centrální diferenci a je velice podobná masce Prewittové. Jediným rozdílem je, že klade větší důraz na příspěvek centrálních pixelů.

Na Sobelovu masku lze také nahlížet jako na 3×3 aproximaci první derivace Gaussovy masky. Jelikož se jedná o ekvivalent nejprve rozostření obrazu pomocí 3×3 aproximace gaussiánu a následné detekci hran s využitím první derivace.

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \tag{10}$$

Velikost gradientu lze získat jako normu gradientů v obou směrech, často se však pro urychlení také používá aproximace pomocí absolutních hodnot, jejíž výpočet přináší značné zrychlení:

$$|g| = \sqrt{(K_x * f)^2 + (K_y * f)^2}$$
$$|g| = |K_x * f| + |K_y * f|\,.$$

3.3 Metody druhé derivace

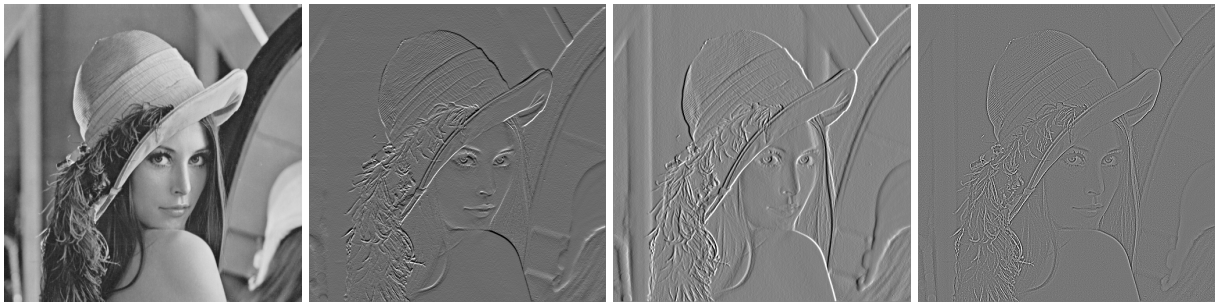
V případě první derivace gradient nabývá pouze jednoho extrému v místě hrany. Druhá derivace průběhu jasu ve směru napříč hranou nabývá dvou extrémů opačného znaménka. Hrana se nachází mezi těmito dvěma extrémy, kde je derivace rovna 0 [11]. V případě diskrétních obrazů je derivace opět nahrazena diferencí:

$$d_x = I(x-1, y) - 2I(x, y) + I(x+1, y)\,,$$
$$d_y = I(x, y-1) - 2I(x, y) + I(x, y+1)\,.$$

Velmi populární operátor druhé derivace je laplacián, definován rovnicí (11), který je možné na obraz aplikovat s využitím konvoluce. Zároveň však tyto metody trápí několik typických problémů, kvůli kterým je jejich použití při detekci hran spíše historické:

1. jsou ještě citlivější na šum v obraze než metody první derivace,
2. produkují 3 pixely silné hrany - průchod nulou, záporný a kladný extrém.

$$K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \tag{11}$$



Obrázek 5: Ukázka aplikace výše popsaných masek na vstupní obraz, Robertsova (druhá zleva), Sobelova (druhá zprava), laplacián (první zprava).

3.4 Cannyho detektor hran

Přestože byl tento detektor navržen již v počátcích počítačového vidění (1986), stal se jednou ze standardních metod pro detekci hran v obraze a je dodnes hojně používán. Výsledkem toho je, že dnes již je v podstatě součástí každé knihovny pro zpracování obrazu.

Cílem Johna F. Cannyho [12] bylo vytvořit algoritmus, který je optimální vzhledem k následujícím kritériím:

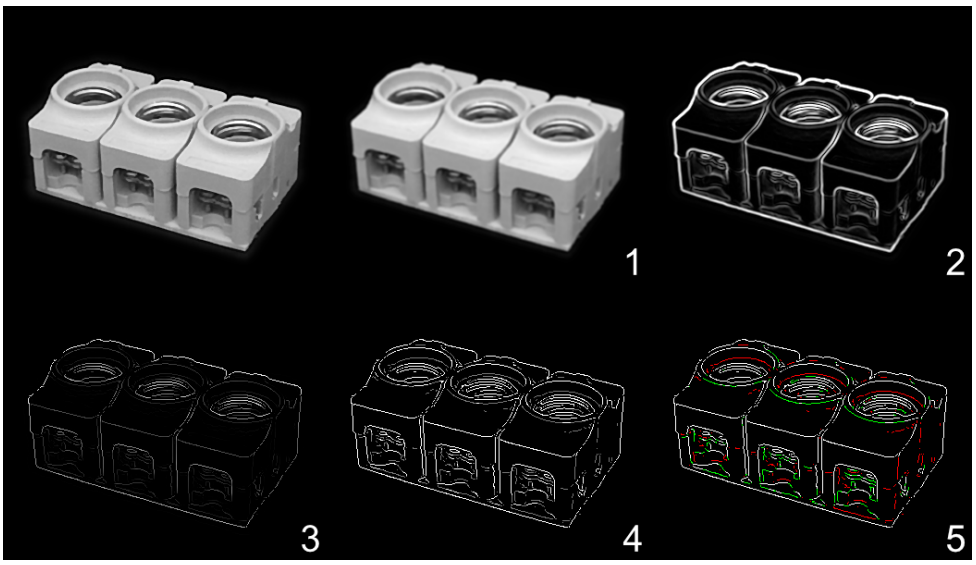
- **Detekce** – pravděpodobnost detekce reálných hran by měla být maximalizována, přičemž pravděpodobnost detekce bodů neobsahujících hranu by měla být minimalizována.

- **Lokalizace** – detekované hrany by měly být co nejvíce blízké reálným hranám.
- **Počet detekcí** – hrany by měly být v obraze detekovány jednoznačně.

3.4.1 Algoritmus detekce hran

Cannyho detektor hran zpracovává vstupní obraz v pěti krocích:

1. **Vyhlazení** – obraz je rozostřen (často s využitím Gaussova operátoru) s cílem snížit šum.
2. **Nalezení gradientů** – detekce gradientů v obraze jednou z výše zmíněných metod, nejčastěji však postačí pouze centrální diference, tu lze vypočítat dle rovnice (4).
3. **Non-maxima suppression** – ponechávají se pouze pixely hran s největší velikostí. To vede ke ztenčení detekovaných gradientů na hrany o velikosti 1 px.
4. **Dvojité prahování** – v obraze jsou ponechány pouze hrany mající velikost větší než t_2.
5. **Sledování hran hysterezí** – výsledné hrany jsou určeny potlačením všech ostatních, které nejsou přímo propojeny s hranami, jež měly ve fázi dvojitého prahování velikost větší než t_1.



Obrázek 6: Vizualizace jednotlivých kroků Cannyho detektoru hran.

3.5 Metody redukce šumu

Existuje mnoho algoritmů redukce šumu v obraze počínaje konvolučními metodami masek různých velikostí, konče filtry, které pracují ve frekvenční doméně. Avšak mezi jedny z nejpoužívanějších nepochybně patří Gaussův filtr.

3.5.1 Gaussův filtr

Gaussův vyhlazovací operátor definuje 2D konvoluční masku, která slouží k redukci šumu a odstranění detailů v obraze. V jistém slova smyslu je podobný uniformnímu filtru (12). Hodnoty Gaussovy masky však reprezentují tvar Gaussovy funkce.

$$K = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{12}$$

Myšlenka Gaussova vyhlazování spočívá v použití 2D distribuce jako funkce bodového rozostření. Jelikož hodnoty jednotlivých pixelů jsou uloženy jako diskrétní čísla, je třeba provést diskrétní aproximaci Gaussovy funkce k vyplnění masky jednotlivými koeficienty. Teoreticky je distribuce Gaussovy funkce všude nenulová, to by však vyžadovalo nekonečně velkou konvoluční masku. Proto jsou v praxi všechny hodnoty ve vzdálenosti větší než 3σ nulové. Zároveň tato vzdálenost definuje velikost výsledné masky.

Často používaná velikost masky je 3×3 a 5×5, definované rovnicí (13), je ale možné ji přizpůsobit v závislosti na dané situaci. V případě větší masky je třeba dbát na to, že lze dosáhnout značné redukce šumu, avšak na úkor větší ztráty detailů.

$$K_{3\times3} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \quad K_{5\times5} = \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}. \tag{13}$$

3.6 Prahování

Výsledkem detekce hran je černobílý obraz, jehož hodnoty pixelů obsahují velikosti gradientů, které v daném místě vznikají. Prahování představuje jednu z nejjednodušších metod segmentace obrazu, jehož použití spočívá v definování prahu t, který udává minimální (maximální) hodnotu pixelu. Celý obraz se následně prochází a pokud je v daném bodě hodnota pixelu menší než definovaný práh t, přepíše se na nulu, dle rovnice (14). V opačném případě se zde tato hodnota buď ponechá, nebo se zapíše jedna, pokud je cílem vytvořit binární obraz. Prahování tedy poskytuje jednoduchý způsob, jak oddělit pozadí od popředí a v případě detekce hran, jak z obrazu odstranit hrany s malou intenzitou, které nejsou v následném zpracování daného obrazu žádané.

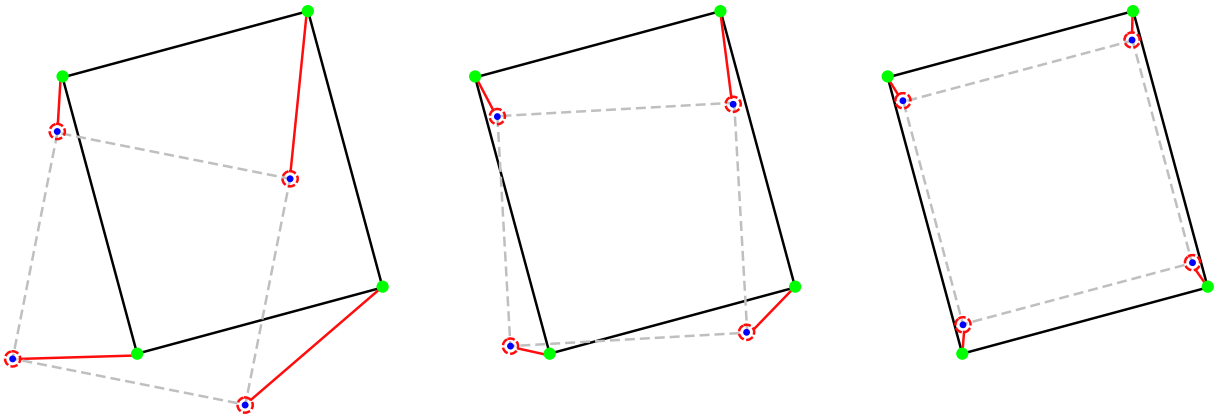
$$f(x,y) = \begin{cases} 1, & \text{pokud } f(x,y) \geq t \\ 0, & \text{jinak} \end{cases}. \tag{14}$$

Výše popsaná metoda využívá pouze jeden práh, který značně omezuje její využití pouze na konkrétní aplikace. Mezi nejčastěji používané alternativy patří metoda automatického určení prahu na základě histogramu.

4 ICP

Iterative closest point (ICP) je široce používaný algoritmus pro geometrické zarovnání 3D modelů v případech, kdy je známý odhad jejich relativní počáteční pózy. Jeho cílem je minimalizovat rozdíl mezi dvěma mračny bodů. Stal se v podstatě dominantní metodou pro zarovnání 3D meshů na základě jejich geometrie (někdy i barvy). Dále je často používán pro registraci výstupu 3D skenerů, modelů kostí apod. Tento algoritmus byl poprvé představen autory Chen, Medioni [13] a Besl a McKay [14].

ICP typicky pracuje s dvěma mračny bodů, kde jedno je neměnné (referenční nebo cílové) a to druhé je transformováno tak, aby co nejlépe odpovídalo tomu referenčnímu. Algoritmus dále iterativně upravuje dané transformace (translace a rotace) s cílem minimalizovat chybovou metriku (typicky se jedná o vzdálenost mezi referenčním a zdrojovým mračnem bodů), kterou může být suma absolutních rozdílů souřadnic jednotlivých párů bodů.



Obrázek 7: Vizualizace průběhu algoritmu ICP.

Dnes již existuje mnoho variant původního ICP [15]. Ty lze klasifikovat do několika kategorií na základě fáze algoritmu, kterou ovlivňují:

1. **selekce** množiny bodů z jednoho nebo obou meshů,
2. **přizpůsobení množiny bodů** k vzorkům jiných meshů (výpočet transformačních matic),
3. **vážení** příslušných párů odpovídajícím způsobem před jejich zarovnáním,
4. **zamítnutí** jednotlivých párů nebo celých množin odlehlých bodů,
5. **přiřazení** chybové metriky jednotlivým párům bodů,
6. **minimalizace chyby** příslušným algoritmem.

5 Použitý dataset

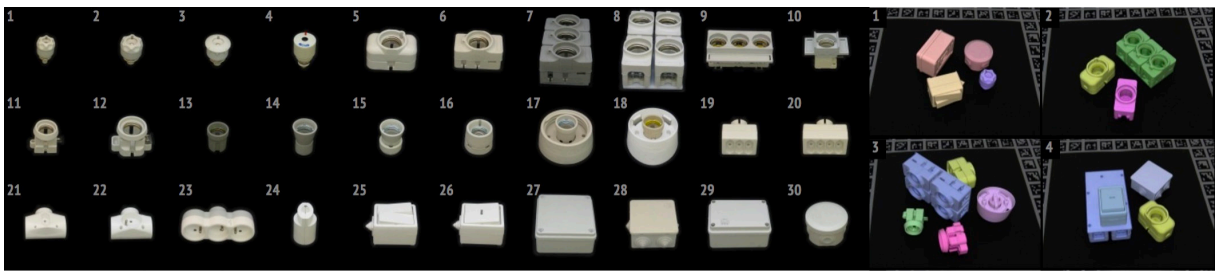
Všechny experimenty a měření byly v rámci této diplomové práce prováděny na veřejně dostupném T-LESS datasetu od T. Hodaně [16]. Tento dataset obsahuje 30 objektů bez textury, mnohdy sobě navzájem podobných svým tvarem, nemajících žádnou výraznou barvu. To jej činí obtížným pro mnoho detekčních algoritmů. Všechny snímky v rámci datasetu jsou k dispozici ze tří různých synchronizovaných senzorů:

- Primesense CARMINE 1.09 (RGB-D),
- Microsoft Kinect v2 (RGB-D),
- Canon IXUS 950 IS (RGB snímky ve vysokém rozlišení).

Každý objekt je definován 1296 templaty ve formátu RGB spolu s 16bitovými hloubkovými mapami. Tyto templaty zachycují objekt z různých úhlů na horní polovině hemisféry, s 10° krokem elevace (od 85° do −85°) a 5° azimutem. Dále dataset poskytuje 3D modely ve formátu .ply, které odpovídají jednotlivým objektům. Ke každému templatu je navíc k dispozici také datový soubor ve formátu .yml obsahující následující parametry:

- `cam_K` – 3×3 matice vnitřních parametrů kamery (uložena po řádcích),
- `cam_R_w2c` – 3×3 rotační matice (uložena po řádcích),
- `cam_t_w2c` – 3×1 translační vektor,
- `elev` – přibližná elevace, ve které byl objekt zachycen,
- `mode` – režim snímání (0 - objekt je postaven kolmo, 1 - objekt je vzhůru nohama).

Mimo výše zmíněné objekty dataset také obsahuje 20 scén pro testování, které jsou opět zaznamenány s využitím všech tří senzorů. V rámci těchto dvaceti scén se jejich složitost velice liší, od jednoduchých až po komplexní scény s velkým počtem objektů a částečnou okluzí. To umožňuje algoritmus dostatečně otestovat ve velkém množství odlišných podmínek.



Obrázek 8: Ukázka dostupných objektů a scén v použitém datasetu [16].

5.1 Úpravy datasetu

I přesto, že tento dataset poskytuje velké množství objektů a scén včetně anotací a vnitřních parametrů kamery pro každý snímek, které jsou uložené v přiložených .yml souborech, je nutné provést několik úprav a dodatečných výpočtů, aby jej bylo možné použít v níže popsané metodě.

5.1.1 Sjednocení velikosti templatů

Všechny templaty objektů v datasetu jsou zachyceny ve stejné vzdálenosti od kamery a umístěny do středu obrázku o velikosti 400×400 px. Jelikož dataset poskytuje objekty různých velikostí, liší se tak i velikost předem definovaných bounding boxů, které definují jejich hranice v každém templatu. Pro správné fungování níže popsané metody (zejména části hashování) je potřeba, aby byly všechny obrázky stejně velké. Proto je nutné projít všechny templaty datasetu hledaných objektů a proporcionálně je zmenšit či zvětšit, aby se vešly do předem dané velikosti obrázku (v rámci této práce je velikost rovna 108×108 px).

5.1.2 Přepočet hodnot hloubkových dat

Po změně velikosti templatu je nutné upravit hodnoty hloubkových dat tak, aby odpovídaly nové vzdálenosti. Pokud byl template zmenšen či zvětšen o faktor s, jedná se o situaci podobnou tomu, kdy byla kamera od objektu vzdálena či přiblížena. Je tedy nutné všechny hodnoty pixelů hloubkových obrazů tímto faktorem vydělit: $I(x, y) = I(x, y)/s$, aby tak došlo k zvětšení, nebo v opačném případě zmenšení hodnoty (vzdálenosti) v hloubkových datech.

5.1.3 Výpočet oblasti, která je templatem pokryta

Po provedení ověření kandidátů je nutné pro výpočet výsledného skóre znát velikost oblasti, kterou objekt v templatu pokrývá. Ta je následně vztažena relativně vzhledem k velikosti obrázku. Lze ji vypočítat s využitím znalosti toho, že objekty jsou v templatech vyobrazeny vůči černému pozadí. Je proto definován práh $t = 40$, kde pokud hodnota pixelu černobílého obrázku je větší než t, oblast templatu a_t se navýší o 1. Takto se projde celý vstupní obraz. Nakonec je výsledná hodnota vztažena relativně vzhledem k velikosti okna pomocí vzorce: $a_{nt} = a_t/(wh)$, kde w a h představují výšku a šířku výsledného obrázku.

5.1.4 Přepočet vnitřních parametrů kamery

V poslední řadě je nutné upravit matici vnitřních parametrů kamery K tak, aby její hodnoty odpovídaly nové velikosti obrázku. Toho lze dosáhnout s využitím rovnice (15).

$$K = \begin{pmatrix} s\ f_x & \gamma & s\ x_0 & 0 \\ 0 & s\ f_y & s\ y_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{15}$$

kde s je faktor změny velikosti, γ zkosení obrazu, f_x, f_y ohnisková vzdálenost osy x, y a x_0, y_0 jsou souřadnice bodu ve středu kamery (v pixelech). V případě, že bounding box daného templatu není čtvercový a template bylo nutné posunout tak, aby co nejefektivněji vyplnil výsledný obraz, je navíc nutné posunout střed kamery. To lze provést pomocí rovnice (16).

$$K = \begin{pmatrix} f_x & \gamma & x_0 - (x_i s) & 0 \\ 0 & f_y & y_0 - (y_i s) & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \tag{16}$$

kde x_i, y_i jsou levé horní souřadnice posunutého bounding boxu.

5.1.5 Generování pomocného souboru `info.yml`

Program po zpracování všech výše uvedených změn generuje nový dataset, včetně obrázků a souboru `info.yml`. Ten rozšiřuje původní soubor o nově vypočítaná data, která dále usnadňují práci při zpracování datasetu. Každému objektu je přiřazen jeden soubor `info.yml`, který obsahuje pole záznamů pro každý template, jejichž struktura vypadá následovně:

- `objId` – id objektu (v případě tohoto datasetu hodnota v rozsahu $1 - 30$),
- `id` – id konkrétního templatu (při parsování je každému templatu přiděleno unikátní id, z předem definovaného čítače),
- `fileName` – jméno souboru,
- `diameter` – průměr reálného objektu (v mm),
- `resizeRatio` – faktor změny velikosti,
- `objBB` – souřadnice a velikost bounding boxu,
- `objArea` – oblast, kterou template pokrývá vztažena vzhledem k velikosti obrázku,
- `minDepth` – minimální hodnota hloubkové mapy, která se v templatu nachází,
- `maxDepth` – maximální hodnota hloubkové mapy, která se v templatu nachází,
- `camera` – objekt, který obsahuje hodnotu elevace, azimutu, rotační matici kamery R, translační vektor t a aktualizovanou matici vnitřních parametrů kamery K.

6 Souvisejici práce

Detekce 3D objektů v obrazech představuje široce zkoumaný problém. Převážnou většinu těchto přístupů lze rozdělit do dvou kategorií:

1. **Metody využívající 3D modelů** – tyto metody využívají převážně CAD 3D modelů nebo hloubkových dat, což je běžné zejména v průmyslových aplikacích.

2. **Metody využívající templatů** – detekce možných póz je v tomto případě limitována počtem templatů objektu. Zároveň její složitost značně narůstá se zvyšujícím se počtem templatů. Bývá často používaná pro detekci aproximované pózy, která je následně upřesněna s využitím odpovídajících 3D modelů.

V nedávné práci [17] [18] byla představena efektivní metoda template matchingu, kde namísto porovnávání celých obrázků je každý template objektu reprezentován body příznaků v několika modalitách (konkrétně 3D normály hloubkových obrazů a orientace gradientů). Hodnoty jednotlivých příznaků jsou následně kvantizovány a reprezentovány pomocí bitových vektorů, což umožňuje rychlé porovnávání s využitím bitových operací. Nalezený template poté poskytuje výchozí 3D pózu objektu, která je následně upřesněna s využitím ICP algoritmu [14]. S datovými strukturami optimalizovanými pro rychlý přístup k paměti a optimalizovaným algoritmem, jenž využívá speciálních vektorových instrukcí (SSE), je tento algoritmus schopen detekce objektů v reálném čase. Rychlost této metody bohužel (jako u mnoha dalších) silně degraduje při větším počtu templatů. Její složitost je lineární vůči počtu templatů v databázi.

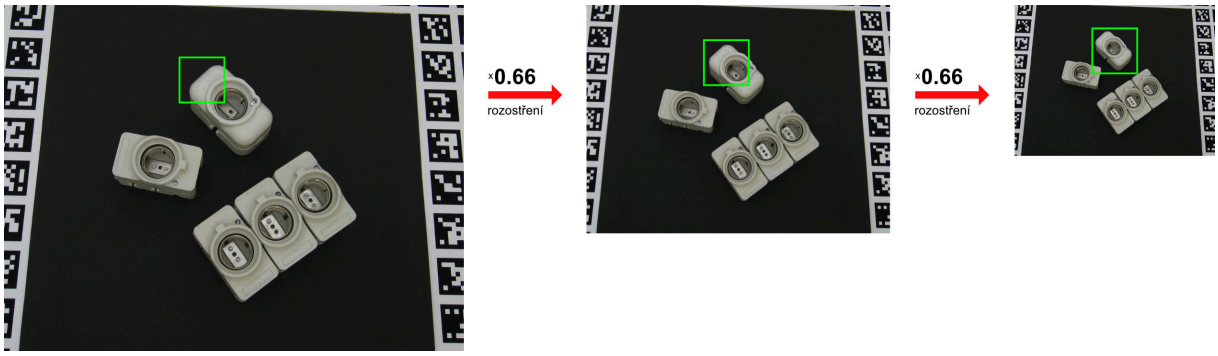
Jedna z dalších metod [19] popisuje detekci s využitím pouze 3D modelů objektů. Ta definuje globální deskriptor objektu, který se skládá ze všech možných párů 3D bodů modelu, jež jsou následně uloženy do hashovací tabulky. Při detekci jsou vybrány páry 3D bodů ze scény, pro něž se následně hlasuje. Množina bodů s nejvíce hlasy definuje výchozí pózu objektu, která může být následně upřesněna s využitím ICP. V [20] byla tato metoda rozšířena o barevné informace bodů, které jsou uloženy spolu s jejich orientací v hashovací tabulce.

Rychlost a účinnost výše zmíněných metod přímo závisí na složitosti testovací scény, což může limitovat jejich reálné použití. Přesto však využití několika předem vyselektovaných bodů příznaků ve více modalitách (3D normály, hloubková data, orientace gradientů a barevné informace v prostoru HSV) inspirovalo fázi ověření templatů v níže popsané detekční kaskádě. Zároveň princip hashovacích tabulek zmíněných v [20] spolu s procesem hlasování, kde bylo hashování založeno na základě vzdálenosti a orientace k nejbližší hraně (jehož implementace byla také nejprve použita v [21]), byl převzat a jeho varianta je použita v evaluační kaskádě ve fázi výběru kandidátů.

7 Detekce objektů

Jak již bylo zmíněno v úvodu, dosavadní metody detekce objektů bez textury v převážné míře provádějí detekci s využitím posuvného okénka, většinou o velikosti templatu, které se po scéně posouvá s předem definovaným krokem. Poté se počítá míra podobnosti mezi obsahem posuvného okénka a templatem. Pokud je tato podobnost větší než definovaný práh t, můžeme klasifikovat okno jako obsahující daný template.

Tento postup bývá často také doplněn o metodu obrazových pyramid, kdy po každém průchodu okna scénou je scéna zmenšena o daný faktor f. Zpravidla je obraz také mírně rozostřen s cílem minimalizovat artefakty a scéna je procházena znovu posuvným oknem. Tento princip umožňuje nalézt hledaný objekt napříč několika měřítky. Jedná se však o výpočetně velmi náročnou metodu, která klade velký důraz na implementaci pro zajištění výpočtu téměř v reálném čase. Není tedy vhodná pro použití v situaci, kdy chceme najít velké množství odlišných objektů, kde s narůstajícím počtem naučených templatů roste časová náročnost.



Obrázek 9: Metoda obrazových pyramid.

Metoda, která byla v rámci této diplomové práce zpracována, se této časové náročnosti snaží vyhnout implementací kaskádové evaluace oken spolu s metodou obrazových pyramid s 4 úrovněmi nahoru i dolů a faktorem zmenšení $q = 1{,}25$. V prvním kroku kaskády dochází k eliminaci velkého množství pozic oken aplikací jednoduché kontroly salience[2] (objektivity). Pro každé zbylé okno je rychle a efektivně vybrána podmnožina nejlepších kandidátů ze všech templatů, metodou hlasování s využitím hashovací tabulky naučené na množině bodů rozvržených v předem definované mřížce. Využití hashovací tabulky zaručuje sub-lineární časovou náročnost v závislosti na počtu hledaných objektů, jelikož pro každé okno je vždy vybrán pouze až N počet kandidátů. Jednotlivé podmnožiny kandidátů jsou následně porovnávány na základě předem získaných příznaků (povrchové normály, orientace gradientů, hloubkový rozdíl a barva v prostoru HSV). Každému templatu je na začátku přiřazena jeho orientace v 3D prostoru a vzdálenost od kamery. Díky tomu je po nalezení správného kandidáta již známa jeho přibližná poloha. V po-

[2]Salience - soubor metod, které se snaží oddělit sledované objekty od pozadí scény stejným způsobem jako to dělá lidské oko.

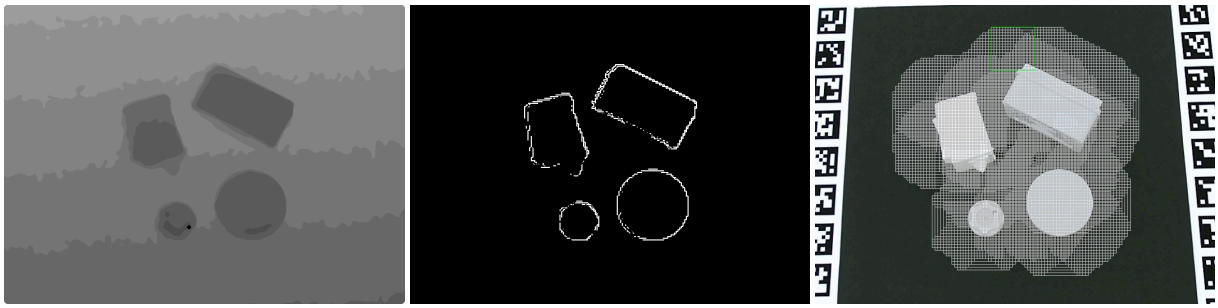
sledním kroku kaskády již dochází k přesnému určení pózy objektů s využitím jejich přibližné polohy, posazením odpovídajícího 3D objektu do vstupní hloubkové mapy a stochastických metod.

Obrázek 10: Vizualizace evaluační kaskády.

Využití kaskádového zpracování umožňuje nejprve vyfiltrovat velké množství oken a kandidátů s využitím výpočetně méně náročnějších úkonů a provést přesnější, avšak časově náročnější, výpočty již na zlomku původního počtu. To kladně přispívá k redukci časové náročnosti celkového výpočtu. V následujících několika sekcích budou blíže popsány jednotlivé části evaluační kaskády a jejich využití v detekčním algoritmu.

7.1 Předfiltrování pozic oken (detekce objektivity)

Cílem první části evaluační kaskády je rychle odstranit velké množství oken, v nichž se nenachází žádný objekt. To zahrnuje test objektivity, tedy určení, s jakou pravděpodobností se v testovaném okně nachází objekt. Jedná se v podstatě o dvoutřídní klasifikátor rozlišující mezi třídou pozadí a třídou objektů, která zahrnuje všechny hledané objekty. Jednou z možných metod pro detekci objektivity je použití binarizovaných normovaných gradientů (BING) popsaných v [22]. Ta je díky použití binárních operací aplikovaných na obrázky velikosti 8×8 popisující orientace gradientů velice rychlá, avšak pro účely této diplomové práce až příliš komplikovaná.



Obrázek 11: Výsledek předfiltrování oken aplikací detekce objektivity (zleva: vstupní hloubkový obraz, edgely v obraze, množina oken obsahujících objekt).

Metoda, která byla v této diplomové práci použita, je založena na sledování počtu edgelů, tedy pixelů, které v hloubkových obrazech představují hranu. Algoritmus odpovídá klasické metodě posuvného okna s velikostí rovné velikosti nejmenšího objektu v databázi a krokem posunu $s = 5$. Edgely vznikají na místech, kde výsledek aplikace Sobelova operátoru je větší než práh θ_o, který je roven 30 % fyzického průměru nejmenšího objektu v databázi (108×108 px). Okno je klasifikováno jako obsahující objekt, pokud má alespoň 30 % edgelů jako objekt, který

jich má nejméně. Tuto hodnotu lze získat ve fázi trénování aplikací stejných operací na jednotlivé templaty objektů. Díky tomuto nastavení je metoda tolerantní vůči částečnému překrytí, ale přesto dostatečně robustní tak, že dokáže eliminovat až 90 % oken v závislosti na počtu objektů a nepořádku ve scéně. Okna, která projdou testem objektivity, jsou poslána dále v kaskádě pro další zpracování.

7.1.1 Urychlení výpočtu s využitím integrálního obrazu

Jelikož detekce objektivity pracuje s binárním obrazem, který obsahuje informace o jednotlivých edgelech, a pro každé okno je nutné spočítat jejich počet, nabízí se možnost použití integrálního obrazu.

Integrální obraz se často používá jako rychlá a efektivní metoda výpočtu sumy hodnot pixelů v obraze, či jakéhokoliv čtverce umístěného ve scéně. Jednotlivé pixely integrálního obrazu obsahují součet hodnot doleva a nahoru [23], tedy pravý dolní pixel představuje součet všech hodnot pixelů v celém obraze. Integrální obraz lze získat pomocí rovnice (17).

$$I_{\Sigma}(x,y) = \sum_{i=0}^{i \leq y} \sum_{j=0}^{j \leq x} I(x,y)\,, \tag{17}$$

kde $I(x,y)$ představuje hodnotu pixelu vstupního obrazu na souřadnicích x, y. Pak už je velice jednoduché získat sumu hodnot pixelů konkrétního čtverce vymezeného body A, B, C, D s využitím rovnice (18).

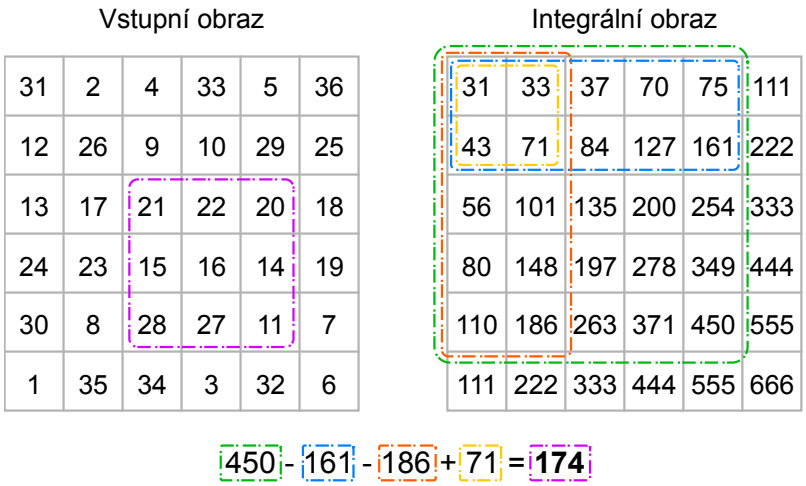
$$\Sigma_E = I_{\Sigma}(A) - I_{\Sigma}(B) - I_{\Sigma}(C) + I_{\Sigma}(D)\,. \tag{18}$$

Využití integrálního obrazu umožňuje dle vztahu (17) provést jeho předpočítání a následně se na počet jednotlivých edgelů v hledaných oknech dotazovat vztahem (18). Tímto způsobem lze předejít nutnosti opakovaného počítání sumy, která je nahrazena jednoduchým výpočtem, čímž se značně sníží časová náročnost celkové detekce objektivity.

7.2 Výběr kandidátů

Úkolem této fáze je rychle identifikovat malou podmnožinu kandidátů pro každé okno, které prošlo detekcí objektivity. Pro jednotlivá okna je následně vybráno až N kandidátů s největší pravděpodobností $P_t(t|w_1)$ (pravděpodobnost, že okno w_1 obsahuje template objektu t). Na tuto proceduru lze také nahlížet jako na vícetřídní klasifikační problém, kde existuje jedna třída pro každý template a žádná pro pozadí.

Princip spočívá ve výběru kandidátů z předem (pro robustnost) natrénovaných hashovacích tabulek, což zaručuje konstantní časovou složitost $O(1)$ v závislosti na počtu natrénovaných templatů. Každá hashovací tabulka $h \in H$ je indexována trénovací množinou měření M_h (získanou na templatu t nebo okně w), která je jiná pro každou tabulku. Jednotlivé buňky tabulky



Obrázek 12: Vizualizace výpočtu oblasti v obraze s využitím integrálního obrazu.

dále obsahují list templatů objektů se stejným klíčem. Pro každé okno w jsou generovány množiny měření M_h pro všechny hashovací tabulky H. Tato měření slouží poté jako klíč k buňce hashovacích tabulek.

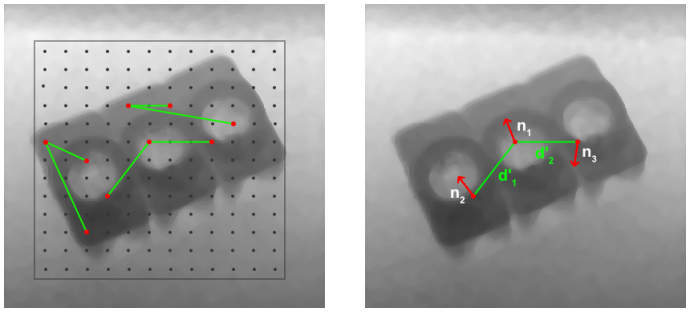
Následně probíhá proces hlasování pro templaty t nacházející se v buňkách pod klíči M_h. Každý template t může získat až $|H|$ hlasů. V tomto případě by všechna měření provedená na všech hashovacích tabulkách ukazovala na buňky obsahující daný template t. Až N templatů s největším počtem hlasů a minimem hlasů v je posláno dále evaluační kaskádou pro další zpracování.

7.2.1 Vytváření hashovacích tabulek

Nejprve se ve fázi trénování přes daný template t umístí pravidelná mřížka velikosti 12×12, která vytváří **144 referenčních bodů**, odkud jsou vzorkovány výsledné $k-tuply$ (množina k referenčních bodů). V případě této diplomové práce byla použita hodnota $k = 3$ (vytvářejí tzv. triplety). Každému tripletu jsou přiřazeny povrchové normály n_k a relativní hloubky d_{k-1}. Ty vytvářejí množinu měření $M_h = (d_2 - d_1, d_3 - d_1, n_1, n_2, n_3)$, která se skládá ze dvou hodnot relativní hloubky a tří povrchových normál. Konkrétní měření M_h poté vytváří klíč hashovací tabulky h. Tento klíč lze následně použít pro uložení daného templatu na odpovídající místo v hashovací tabulce.

Ve fázi detekce objektů, kdy dochází k výběru až N kandidátů pro jednotlivá okna, je proces velice podobný fázi trénování. Znovu je přes dané okno w umístěna pravidelná mřížka a dochází k výpočtu hodnot pod jednotlivými triplety hashovacích tabulek, které tvoří výše zmíněný klíč. Tentokrát je však tento klíč použit k dotazu na templaty, které se pod ním v hashovací tabulce nacházejí. Pro ně je následně spuštěn proces hlasování, kde každý template dostane jeden hlas, pokud se pod daným klíčem vyskytoval. Tímto způsobem se projde všech H tabulek a prvních N

templatů s počtem hlasů vyšším než v je vybráno jako potencionální kandidáti, kteří se v obraze vyskytují. Ti jsou následně ověřeni v další fázi detekční kaskády.

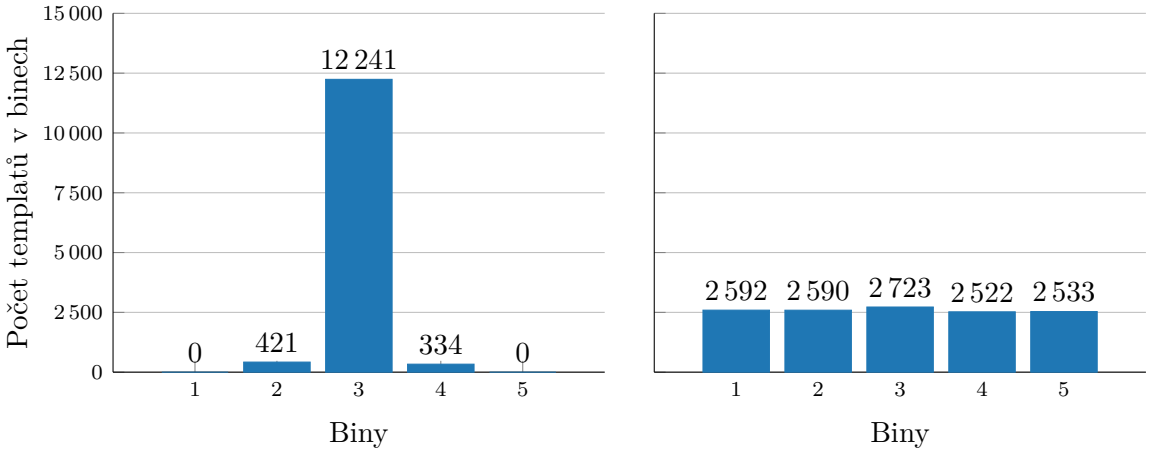


Obrázek 13: Ukázka několika tripletů rozvržených v pravidelné síti 12×12.

7.2.2 Výpočet a kvantizace hloubkových rozdílů

Relativní hloubky $(d_2 - d_1)$ a $(d_3 - d_1)$ jsou kvantizovány do pěti binů (diskrétní hodnoty definované v předem daných intervalech), které tvoří hodnotu klíče tabulky. Přestože teoreticky mohou být tyto hodnoty v rozsahu $[-65365, 65365]$, definování hranic jednotlivých binů jeho rozdělením „hrubou silou" není vhodné.

Rozsahy jednotlivých binů se tudíž liší pro každý triplet (tabulku) a jejich hodnoty se určí ve fázi trénování tak, aby v každém binu byl přibližně stejný počet templatů. Tyto hraniční hodnoty lze získat předběžným výpočtem hloubkových rozdílů na všech templatech tabulky, jejichž výsledkem je množina D. Ta je následně seřazena vzestupně. Dále je nutné zjistit průměrný počet templatů na jeden bin, což lze provést jednoduchým dělením celkového počtu hloubkových rozdílů množiny D počtem binů, v tomto případě pěti: $b = \frac{|D|}{5}$. Nyní již stačí pouze zjistit hodnoty na indexech $1b$, $2b$, $3b$ a $4b$ v množině D, které udávají vnitřní hraniční hodnoty jednotlivých binů. Minimální a maximální relativní hloubka definující hranici prvního a posledního binu je rovna hodnotě prvního a posledního prvku množiny D.



Obrázek 14: Definování hranic binů „hrubou silou" (vlevo) a dynamická definice (vpravo).

7.2.3 Výpočet povrchových normál

Povrchové normály jsou v hloubkových obrazech vyextrahovány pomocí metody popsané v [18]. Ta poskytuje rychlý a robustní způsob jejich odhadu. Nejdříve je nutné definovat kolem každého pixelu obrazu x první stupeň Taylorovy řady hloubkové funkce $D(x)$, tu definuje následující rovnice:

$$D(x + dx) - D(x) = dx^T \nabla D + h.o.t\,.$$

V rámci obdelníku definovaného kolem x o velikosti 5 px na každou stranu, každý posun dx vrací rovnici, která dále omezí hodnotu ∇D a umožní tak získat optimální gradient $\hat{\nabla D}$ ve smyslu metody nejmenších čtverců. Tento gradient poté odpovídá 3D rovině, která prochází třemi body X, X_1 a X_2, ty lze vypočítat pomocí rovnice (19).

$$\begin{aligned} X &= v(x)D(x)\,, \\ X_1 &= v(x + [1,0]^T)(D(x) + [1,0]\hat{\nabla D})\,, \\ X_2 &= v(x + [0,1]^T)(D(x) + [0,1]\hat{\nabla D})\,, \end{aligned} \tag{19}$$

kde $v(x)$ je vektor procházející pixelem x v zorném poli kamery, jenž lze vypočítat s využitím znalosti vnitřních parametrů kamery a rovnice (20).

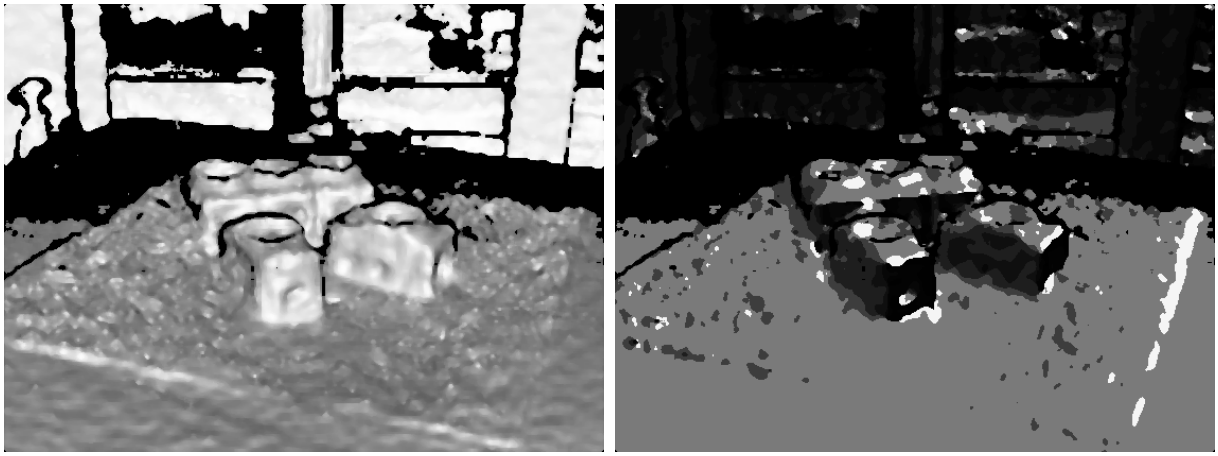
$$\begin{aligned} v_x &= (x - c_x)\ \mathrm{depth}(x,y)/f_x\,, \\ v_y &= (y - c_y)\ \mathrm{depth}(x,y)/f_y\,, \\ v_z &= \mathrm{depth}(x,y)\,. \end{aligned} \tag{20}$$

Normálu 3D roviny, která je projektována bodem x, lze následně získat jako normalizovaný vektorový součin $X_1 - X$ a $X_2 - X$. Tento postup však není robustní kolem překrývajících se kontur, kde již není známa aproximace prvního stupně Taylorovy řady. S využitím bilaterálního filtru jsou ignorovány pixely, jejichž hloubkový rozdíl se středovým pixelem je větší než definovaný práh ($t_d = 100$). Zároveň tento filtr pomáhá eliminovat šum v hloubkových obrazech a přitom stále poskytovat smysluplné odhady povrchových normál.

7.2.4 Kvantizace povrchových normál

Povrchové normály n_1, n_2, n_3 jsou vzorkovány nad hloubkovými obrazy a kvantizovány do 8 diskrétních hodnot na základě [18]. Kvantizace normál spočívá v definování 8 normalizovaných vektorů n_{r_i} umístěných ve středu jednotlivých oktantů horní poloviny sféry. S takto předdefinovanými vektory lze provést skalární součin s povrchovou normálou n_k, který vrací hodnotu rovnou kosinu úhlu svíraného oběma vektory. Jakmile tato funkce najde maximum $\max(n_{r_i} \cdot n_k)$, tak index vektoru n_r odpovídá kvantizované hodnotě.

Tato metoda je však relativně pomalá, a to zejména v případě většího počtu povrchových normál, které je třeba kvantizovat. S využitím předpokladu, že jednotlivé normály vždy směřují



Obrázek 15: Povrchové normály vyextrahované z hloubkového obrazu (vlevo) a jejich kvantizace (vpravo).

ke kameře, a stačí tedy kvantizovat pouze souřadnice x, y daného vektoru, tak jedno z možných urychlení, na úkor menší přesnosti, spočívá v definování 2D vyhledávací tabulky (angl. lookup table) o předem dané velikosti (ta definuje míru přesnosti vyhledávání), která obsahuje jednotlivé diskrétní hodnoty. Ty jsou v tabulce uspořádány do 8 kvadrantů. Pak už jen stačí souřadnice normalizovaného vektoru převést do souřadného systému tabulky, kde 0 reprezentuje její střed. Hledaná kvantizovaná hodnota se poté nachází na souřadnicích table(x, y). Využití vyhledávací tabulky tak umožňuje nahradit složité výpočty jednoduchým vyhledáváním, což kladně přispívá k výsledné rychlosti algoritmu.

```
1 const unsigned char LUT[6][6] = {
2     {2, 2, 2, 3, 3, 3},
3     {1, 2, 2, 3, 3, 4},
4     {1, 1, 1, 4, 4, 4},
5     {8, 8, 8, 5, 5, 5},
6     {8, 7, 7, 6, 6, 5},
7     {7, 7, 7, 6, 6, 6}
8 };
```

Výpis 1: Ukázka vyhledávací tabulky velikosti 6×6.

7.2.5 Generování pozic tripletů

Každé hashovací tabulce je ve fázi trénování přiřazen jeden unikátní triplet, pro něž je následně celá tabulka natrénována nad množinu templatů T. Nad jednotlivými templaty je v pravidelné mřížce daný triplet přiložen a v jednotlivých bodech jsou počítány povrchové normály spolu s relativními hloubkami tvořící klíč konkrétní buňky v hashovací tabulce. Pozice jednotlivých bodů tripletu je však nutné ověřit a zjistit, zda se nacházejí na úrovni objektu daného templatu,

aby tak nedocházelo k neplatným výpočtům. K tomu lze využít RGB obrázků datasetu, kde jsou objekty vyobrazeny vůči černému pozadí, a mohou tak sloužit jako maska. Pokud se některý z bodů tripletu nenachází na úrovni objektu, tento template se do hashovací tabulky neukládá. Pozice tripletů mohou být zvoleny buď náhodně, nebo předem tak, aby:

1. pokryly co největší množství nezávislých měření,
2. vyplnily tabulky co nejrovnoměrněji je možné.

Bohužel zvolení optimální konfigurace patří do třídy problémů NP-úplné. Jedním z možných řešení je vygenerovat $m \times |H|$ tabulek a ponechat ty, které mají největší pokrytí. Hodnota koeficientu m by měla být určena s mírou, jelikož velké hodnoty způsobí to, že většina tripletů bude generována ve středu obrazu, a nedojde tak k dostatečnému pokrytí celé referenční mřížky. Malá hodnota naopak může zapříčinit nedostatečné zaplnění hashovacích tabulek.

S cílem zajistit generování tripletů, které toho o objektech vypovídají co nejvíce je dále zaveden práh $t_a = 30°$, který udává minimální úhel mezi vektory z bodu normály n_1 do bodů normál n_2 a n_3. Dále je mezi těmito body omezena minimální a maximální euklidovská vzdálenost na hodnotu $1{,}2 \leq dist \leq 7{,}2$ (hodnoty jsou udány v souřadném systému referenční mřížky).

7.2.6 Hodnoty parametrů při implementaci

Pro všechny výsledky zmíněné v tomto dokumentu bylo použito $|H| = 100$ hashovacích tabulek s koeficientem trénování $m = 15$, minimálním počtem hlasů $v = 3$ a maximálním počtem kandidátů poslaných dále kaskádou $N = 100$.

7.2.7 Implementace hashovací tabulky

Při implementaci hashovací tabulky v jazyce C++ se nabízí hned několik možností. Níže jsou uvedeny jedny z nejčastějších použití:

- `std::unordered_map` – jedná se o asociativní kontejner, který je součástí standardních knihoven jazyka C++. Obsahuje páry unikátních klíčů, kterým přísluší daná hodnota. Zároveň zajišťuje při vkládání, vyhledávání a mazání záznamů konstantní časovou složitost v průměrném čase. Její rychlost však velmi závisí na definování správné hashovací funkce a ani tehdy nelze zaručit časovou složitost $O(1)$ při náhodném přístupu. To může negativně ovlivnit rychlost běhu algoritmu ve fázi výběru kandidátů.
- `std::vector` – v tomto případě se jedná o sekvenční kontejner, který zapouzdřuje pole dynamické velikosti. Opět je součástí standardních knihoven. Ve srovnání s `std::unordered_map` `std::vector` zaručuje složitost O(1) při náhodném přístupu. Avšak na druhou stranu neumožňuje vytvářet vlastní klíče, a je tak na programátorovi, aby si napsal vlastní převodní tabulku. To většinou vyžaduje inicializaci vektoru na maximální možnou velikost

největšího klíče, což zvyšuje paměťovou náročnost. Přesto však ve většině případů překonává `std::unordered_map` ve všech směrech.

- **Externí knihovny** – poslední možností je využití jedné z mnoha externích knihoven, které se snaží vylepšit nedostatky `std::unordered_map` efektivnější implementací. Mezi ty nejznámější patří: `tsl::hopscotch_map`, `google::dense_hash_map`, `tsl::sparse_map` a `tsl::robin_map`.

Všechny výše zmíněné varianty byly důkladně nastudovány a otestovány. V případě `std::vector` byla použita funkce, která mapovala jednotlivé hodnoty klíčů hashovací tabulky na index v poli pomocí bitových operací. Z výše zmíněných knihoven byla použita `tsl::hopscotch_map`, převážně vzhledem k její jednoduchosti, přímé náhradě za `std::unordered_map` (využívá stejné API) a faktu, že se jedná pouze o header-only knihovnu. U obou hashovacích tabulek byl použit vlastní klíč, který odpovídá výše popsané struktuře, spolu s vlastní implementací hashovací funkce.

Vzhledem k tomu, že se ve fázi trénování a detekce využívá pouze operací vyhledávání a vkládání, byla rychlost ověřena pouze na nich. Testováno bylo náhodné vyhledávání a vkládání 10^6 a 10^7 prvků spolu s reálným použitím při detekci tří objektů v testovací scéně, kde docházelo k vyhledávání kandidátů dle zadaného klíče v jedné ze 100 hashovacích tabulek. Výsledky měření lze pozorovat v následující tabulce.

Tabulka 1: Porovnání rychlosti odlišných implementací hashovací tabulky.

	Vyhledávání		**Vkládání**		**Detekce**
Počet prvků	10^6	10^7	10^6	10^7	10^6 - 10^7
`std::unordered_map` [ms]	143	1343	190	1601	813
`std::vector` [ms]	**883**	**821**	**342**	**1242**	**345**
`tsl::hopscotch_map` [ms]	105	1016	156	1356	463

Z výše uvedené tabulky lze pozorovat, že obě dvě další varianty jsou ve všech případech rychlejší než `std::unordered_map`. Externí knihovna v podobě `tsl::hopscotch_map` si vede trochu lépe, ale nic z toho se nemůže vyrovnat rychlosti vektoru, který byl nakonec použit v rámci implementace.

7.3 Porovnání templatů

Předposlední fáze evaluační kaskády již odpovídá klasickým metodám template matchingu, kdy se jednotliví kandidáti přikládají na pozici okna ve scéně a zjišťuje se jejich podobnost. Díky

filtrování v předchozí fázi kaskády každé okno, které prošlo detekcí objektovosti, obsahuje maximálně N kandidátů. To zajišťuje konstantní časovou složitost vzhledem k počtu hledaných objektů. Jelikož byli jednotliví kandidáti vybráni již ve fázi hashování, lze tuto část kaskády chápat jako N odlišných dvoutřídních klasifikátorů rozlišujících mezi objektem reprezentovaným templatem a pozadím.

Verifikační proces této fáze se skládá z pěti po sobě jdoucích testů. Ty vyhodnocují dané okno na základě různých předpokladů v předem definovaných referenčních bodech. Mezi tyto testy patří:

1. test velikosti objektu vzhledem k sledované vzdálenosti,
2. porovnání povrchových normál,
3. porovnání orientace gradientů,
4. test rozdílů hloubek s využitím hloubkové mapy,
5. porovnání barev pixelů v prostoru HSV.

Výše zmíněné testy jsou seřazeny na základě jejich výpočetní náročnosti. Pokud testovaný template některým z těchto testů neprojde, další testy se již nevyhodnocují a daný template je klasifikován jako neplatný. Obdobně jako v případě hashování, i tady se hodnoty jednotlivých testů v referenčních bodech předpočítávají pro každý template před spuštěním detekce s cílem urychlit běh algoritmu.

7.3.1 Generování referenčních bodů

Jednotlivé testy nepočítají podobnost na celém okně po jednotlivých pixelech s využitím některé z korelačních metod popsaných v 2, ale v předem určených referenčních bodech. To znovu snižuje celkovou časovou složitost algoritmu při zachování dostačující přesnosti, jelikož není nutné zpracovávat všechny pixely templatu, ale výpočet je omezen pouze na několik klíčových bodů. V případě této diplomové práce je generováno **100 referenčních bodů** na hranách (siluetách) a stabilních místech povrchu objektu.

Referenční body jsou vyextrahovány ve fázi trénování nezávisle pro každý template. Jednou z možných metod je aplikace Cannyho detektoru hran s následným prahováním s cílem získat binární obraz, ve kterém jsou vyobrazeny pouze hrany objektu. Poté jsou získány souřadnice všech bílých bodů v obraze, představujících hrany, ze kterých je náhodně vybráno 100 referenčních bodů.

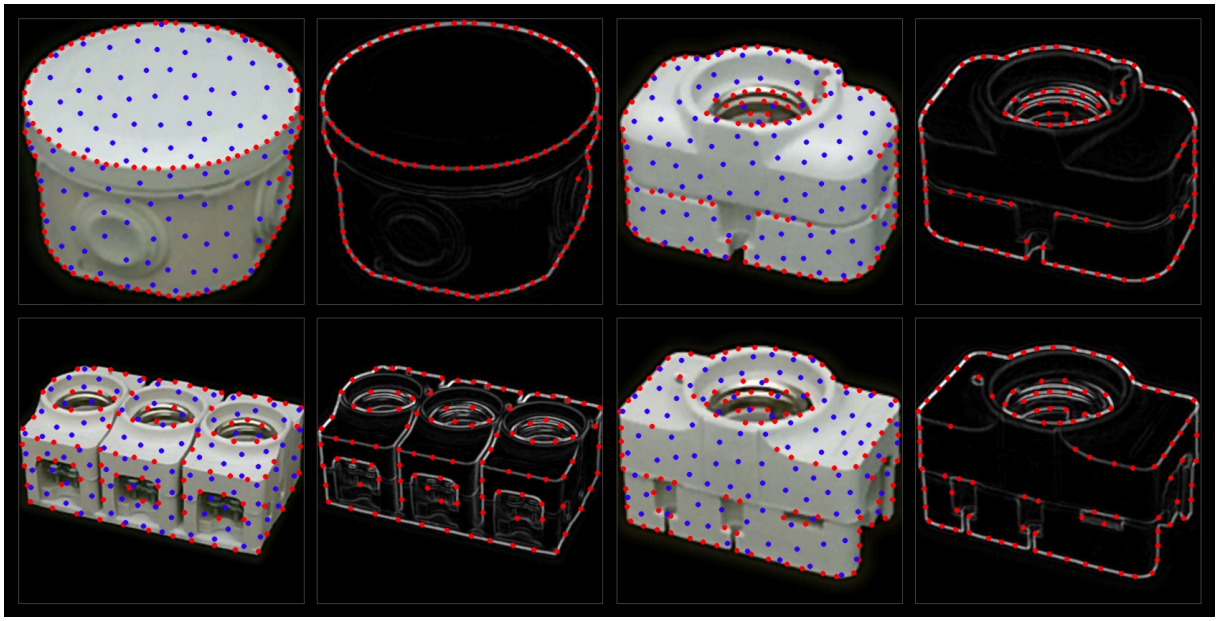
V případě extrakce bodů, které se nacházejí na stabilních místech povrchu objektu, tedy místech relativně neměnných, dále od hran, je nejprve na template aplikován Sobelův operátor s následným prahováním. Zde již není snaha získat přesné pozice hran, ale spíše se hranám vyhnout. Z toho důvodu byl použit Sobelův operátor namísto Cannyho detektoru hran, který

zachytí větší oblast gradientů, tedy pozic s nestabilním povrchem. Po získání těchto bodů je opět procházen celý obraz, kde jsou tentokrát vybírány body, které se nacházejí uvnitř objektu, ale ne na jeho hranách. Z této množiny je nakonec náhodně vybrána podmnožina 100 stabilních referenčních bodů.

Výše uvedený postup však nemůže zaručit **rovnoměrné rozdělení bodů** napříč hranami objektu. Alternativním řešením je použití metody popsané v [17]. Ta nejprve na vstupní template aplikuje Sobelův operátor pro detekci hran, které vytváří množinu bodů E. Ta je seřazena dle jejich intenzity sestupně. Dále je definována konstanta k, která představuje minimální vzdálenost mezi všemi body. Seznam seřazených bodů se poté prochází od prvního prvku, kde jsou odstraněny všechny body, které s ním mají menší než definovanou minimální euklidovskou vzdálenost k. Tu lze vypočítat s využitím rovnice (21).

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}\,. \tag{21}$$

Takto algoritmus pokračuje, dokud v seznamu nezůstanou pouze body splňující podmínku minimální vzdálenosti. Pokud je velikost výsledné množiny menší, či naopak větší než hledaná hodnota, konstantu k je možné zmenšit, respektive zvětšit o definovaný faktor k_d, a celý algoritmus spustit znovu.



Obrázek 16: Ukázka 100 referenčních bodů generovaných uniformně na různých templatech (modrá - stabilní body, červená - body na hranách).

Výchozí hodnoty konstant k a k_d je nutné určit experimentem, jelikož velmi závisí na použitém datasetu. V případě této práce je $k_d = 0{,}5$ a k rovna poměru počtu všech zpracovávaných bodů vůči požadovanému počtu výsledných bodů. Tato heuristika je rozumná, jelikož kontura siluety objektu je většinou 1 px široká hrana, takže tato výchozí hodnota jednoduše definuje ma-

ximální možnou vzdálenost mezi dvěma body, pokud by tyto body byly uniformně rozprostřeny na ideální 1 px široké siluetě.

Výsledkem je účinná metoda, která zajišťuje výběr referenčních bodů, které jsou robustní a zároveň téměr uniformě rozloženy na siluetě objektu. Algoritmus napsaný v jazyce C++ je přiložen níže.

```
1  uniformFeaturePoints(const std::vector<std::pair<cv::Point, uchar>> &points,
       uint count, std::vector<cv::Point> &scattered) {
2          // Definování počáteční minimální vzdálenosti
3          float minDst = points.size() / criteria->featurePointsCount;
4
5          // Postupné snižování minimální vzdálenosti
6          while (scattered.size() < count) {
7              scattered.clear();
8              minDst -= 0.5f;
9
10             // Výpočet euklidovské vzdálenosti mezi všemi body
11             for (size_t k = 0; k < points.size(); ++k) {
12                 bool skip = false;
13                 const size_t edgePointsSize = scattered.size();
14                 // Přeskočení výpočtu pro body, s kterými již existuje vzdá
                       lenost menší než minDst
15                 for (size_t j = 0; j < edgePointsSize; ++j) {
16                     if (cv::norm(points[k].first - scattered[j]) < minDst) {
17                         skip = true;
18                         break;
19                     }
20                 }
21                 if (skip) { continue; }
22                 scattered.push_back(points[k].first);
23             }
24         }
25
26         // Úprava velikosti nalezených bodů na požadovaný počet
27         scattered.resize(count);
28     }
29 }
```

Výpis 2: Algoritmus pro výběr uniformně rozprostřených referenčních bodů.

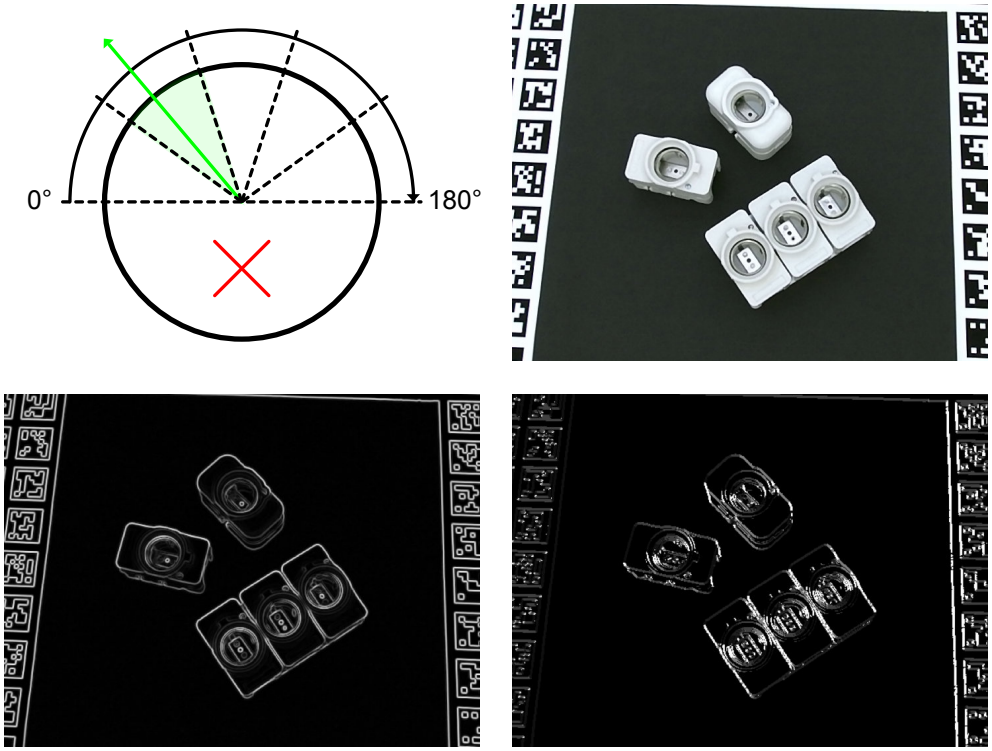
7.3.2 Test velikosti objektů vzhledem k pozorované vzdálenosti

Jedná se o první a zároveň výpočetně nejméně náročný test, který sleduje, zda velikost objektu odpovídá jeho vzdálenosti naměřené v hloubkové mapě. Objekt je očekáván na vzdálenosti Z_e vypočtené jako $Z_e = Z_t s$, kde s odpovídá faktoru aktuálního měřítka obrazové pyramidy a Z_t je natrénovaná (průměrná) vzdálenost objektu v templatu. Pokud je naměřená hloubka Z_w uvnitř intervalu $\langle Z_e/t_d;\ Z_e t_d\rangle$, kde t_d je předem definovaný práh (v případě této práce $t_d = 0{,}85$), hloubka Z_w je považována za správnou, v opačném případě test selže. Pozorovaná hloubka je kontrolována pouze ve středu obrazu v okolí 5×5px.

7.3.3 Test povrchových normál a orientací gradientů

Povrchové normály jsou z hloubkových obrazů extrahovány a následně kvantizovány stejným způsobem, jako bylo popsáno ve fázi hashování v sekci 7.2.3. Orientace gradientů jsou získány z obrazů stupně šedi provedením derivace v daném bodě ve směru x a y pomocí rovnice (4).

Dosazením hodnot d_x, d_y do funkce $\mathrm{arctg}(d_y/d_x)$ lze získat úhel, který gradient svírá s kladnou osou x. Tento úhel je následně kvantizován do 5 binů rozdělením intervalu 0° - 180° do pěti částí [18]. Využití pouze orientace gradientů a ne jeho velikosti činí detekci robustní vůči změnám kontrastu. Kvantizace pouze v jedné polovině kružnice zároveň zajišťuje to, že orientace gradientu nebude ovlivněna tím, zda je objekt vyobrazen vůči světlému či tmavému pozadí.



Obrázek 17: Kvantizace orientací gradientů vstupní scény (vpravo nahoře), výsledek po detekci hran (vlevo dole) a kvantizované hrany (vpravo dole).

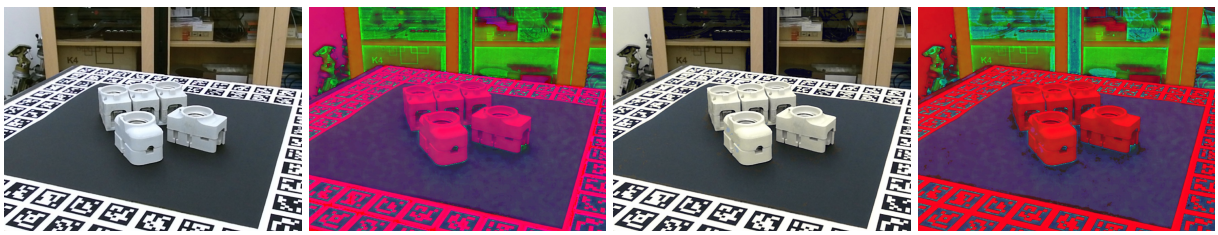
V případě obou testů jsou kvantizované hodnoty porovnávány pouze na předem definovaných referenčních bodech. Pro test orientace gradientů se využívá bodů generovaných na hranách objektu. Naopak povrchové normály využívají stabilních bodů. Při porovnávání jsou všechny referenční body jednotlivých templatů posunuty do souřadného systému aktuálně zpracovávaného okna.

7.3.4 Test rozdílů hloubek a porovnání barev pixelů

Oba testy využívají stabilních referenčních bodů extrahovaných pro povrchové normály. Test rozdílů hloubek pro každý referenční bod porovnává rozdíl d mezi hloubkou v templatu t a hloubkou v okně w. Test projde, pokud je výsledná hloubka $|d - d_m| < kD$, kde d_m je medián rozdílů hloubek napříč všemi referenčními body, D průměr fyzického objektu a k konstanta (v případě této práce je $k = 0{,}5$).

Barvy jednotlivých pixelů referenčních bodů jsou porovnávány v prostoru HSV [17], kde se využívá pouze složky hue, která činí celý test robustní vůči změnám osvětlení. Zjišťuje se, kolik hodnot odstínů jednotlivých pixelů mezi templatem a daným oknem má očekávanou barvu. Pixel má očekávanou barvu, pokud absolutní rozdíl hodnoty hue (modulo π) mezi templatem a oknem je menší než předem definovaný práh t_m.

V případě porovnávání je však stále nutné ošetřit bílé a černé objekty před výpočtem samotného rozdílu. Jelikož černá a bílá barva není pokryta složkou hue, je nutné je mapovat do hue hodnoty podobných barev: černá na modrou a bílá na žlutou. Tento proces je uskutečněn kontrolou hodnoty value a saturation. Pokud je value menší než práh t_v, hodnota hue je mapována na žlutou barvu. Pokud saturation je větší než práh t_s, je hodnota hue nastavena na žlutou barvu. V případě této práce byly hodnoty $t_m = 5$, $t_v = 30$ a $t_s = 40$.



Obrázek 18: Ukázka normalizace HSV, vstup RGB (první zleva), po převodu do HSV (druhá zleva), normalizované HSV v RGB (třetí zleva), normalizované HSV (čtvrté zleva).

7.3.5 Výsledné skóre

Testy 2−5 zmíněné v předchozích několika sekcích projdou, pokud je počet shodných referenčních bodů v malé oblasti kolem zkoumaného bodu (v tomto případě 5×5 s cílem kompenzovat krok posuvného okna $s = 5$ ve fázi detekce objektivity), alespoň 55 %.

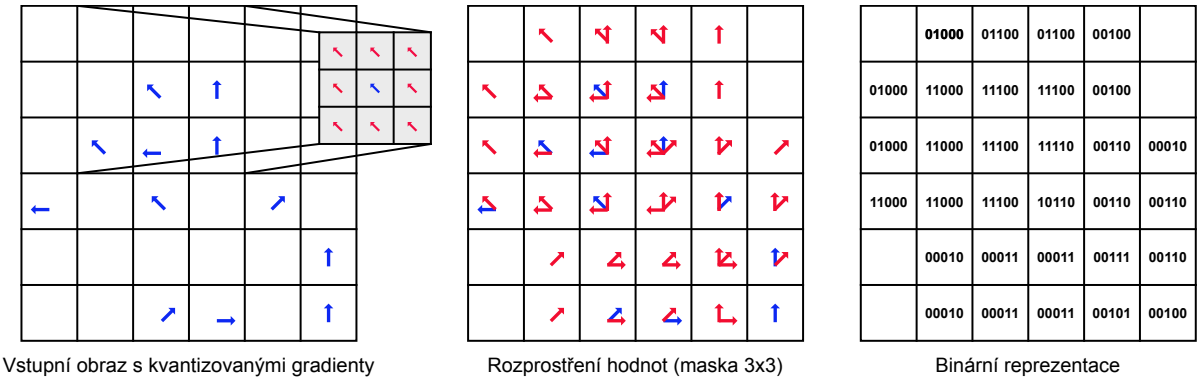
Každému kandidátovi, který projde všemi testy, je přiřazeno výsledné skóre m vypočtené dle vztahu (22), kde C_i je zlomek shodných bodů v testech $2-5$.

$$m = \sum_{i \in \{2,\dots,5\}} C_i \,. \tag{22}$$

7.3.6 Urychlení testů s využitím response map

Aby se zabránilo porovnávání kvantizovaných hodnot (v případě testu povrchových normál a gradientů) každého pixelu v definované oblasti kolem referenčního bodu pokaždé, kdy je nutné vyhodnotit nový template, je zavedena nová binární reprezentace $\mathcal{T}$ kvantizovaných hodnot v předem dané oblasti kolem každého bodu v obraze.

Výpočet $\mathcal{T}$ je znázorněn na obrázku 19. Nejprve jsou nad jednotlivými obrazy vypočítány požadované vlastnosti (povrchové normály a orientace gradientů), které jsou následně kvantizovány do malého množství hodnot, jak již bylo zmíněno v sekcích 7.2.3 a 7.3.3. Maticová reprezentace takto kvantizovaných hodnot umožňuje „rozprostřít" jednotlivé hodnoty vstupního obrazu kolem každého bodu v okolí o velikosti m pro získání nové reprezentace originálního obrazu. Z důvodů vyšší účinnosti jsou všechny možné kombinace kvantizovaných hodnot vstupního obrazu I reprezentovány binárním řetězcem, kde každý bit tohoto řetězce odpovídá jedné kvantizované hodnotě a je roven 1, pokud se daná hodnota vyskytuje v okolí o velikosti m.



Obrázek 19: Vizualizace výpočtu response map s využitím konvoluce.

$\mathcal{T}$ lze vypočítat velice efektivně s využitím konvoluce. Nejprve jsou vypočítány mapy kvantizovaných povrchových normál a orientací gradientů, kde každý pixel odpovídá nenulové hodnotě (mocnina dvou), pokud se na daném místě nachází hledaná orientace, a nule, pokud je orientace neplatná (nesprávná hloubka nebo malá velikost hrany). Následně je na tyto mapy aplikována konvoluce o velikosti masky $m = 5$ (velikost okolí, ve kterém jsou hodnoty porovnávány ve výše zmíněných testech). Na všechny hodnoty, jež se v masce nacházejí, je aplikována binární operace `OR` s centrálním pixelem. Výsledkem této operace je mapa kvantizovaných hodnot, kde binární reprezentace každého pixelu obsahuje všechny orientace, které se nacházejí v jeho okolí m.

Při porovnávání už tudíž není nutné procházet jednotlivě okolí každého referenčního bodu a hledat shodu. Stačí pouze provést binární operaci `AND` s natrénovanou hodnotou aktuálně porovnávaného templatu a hodnotou mapy $\mathcal{T}$ v daném bodě. Pokud je výsledek této operace větší než 0, znamená to, že se v okolí o velikosti m daného pixelu nachází hledaná hodnota. V opačném případě, kdy je výsledek operace roven 0, nebyla v binárním řetězci nalezena žádná shoda a daná orientace se v tomto bodě ani v jeho okolí nenachází.

Přestože předpočítání $\mathcal{T}$ vyžaduje další nároky na výpočetní kapacitu, její použití přináší několikanásobné zrychlení detekčního algoritmu, a to převážně v případě složitějších scén, kdy je ve fázi výběru kandidátů pomocí hashovacích tabulek vybrán větší počet, jenž je následně nutné ověřit. Zároveň je možné vyhnout se kontrolám, zda se daný referenční bod nenachází mimo scénu, jelikož v případě použití původní metody nelze zaručit, že velikost prohledávaného okolí již nezasahuje mimo hranice scény. S využitím response map se již žádné okolí neprohledává, ale používá se pouze hodnota v daném referenčním bodě, jehož pozice je vždy uvnitř daného okna.

Tabulka 2: Porovnání rychlosti ověření kandidátů s využitím response map.

	Scéna 1	**Scéna 2**	**Scéna 3**
Počet kandidátů ve scéně	**173229**	**751096**	**1545466**
Původní implementace [ms]	946	3603	7430
Využití response map [ms]	108	185	291

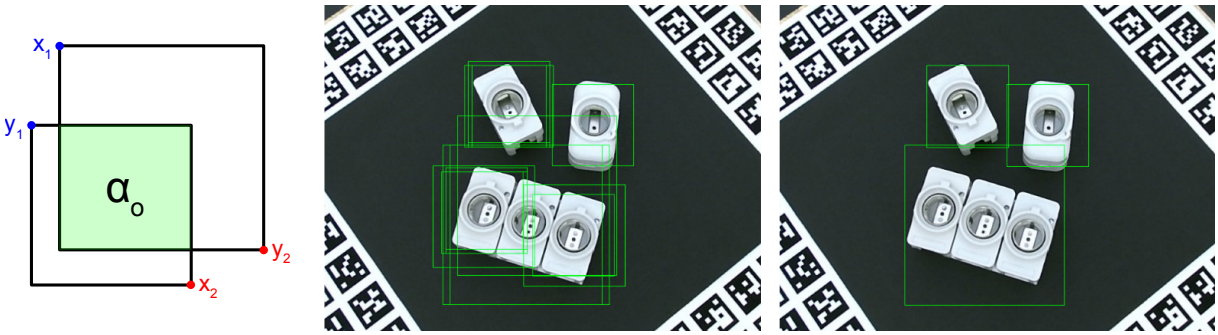
Výše uvedená tabulka znázorňuje porovnání rychlosti detekce s využitím obou metod. Zde byly otestovány 3 různě složité scény s odlišným počtem kandidátů při detekci 3 objektů napříč 9 úrovněmi obrazové pyramidy. V evaluační kaskádě byly response mapy použity v případě dvou testů - porovnání povrchových normál a orientací gradientů. Lze pozorovat, že jednoduchý logický součin, který zde nahradil několik cyklů, jež prohledávaly okolí kolem daného referenčního bodu, několikanásobně zrychluje detekční algoritmus.

Nakonec je důležité zmínit, že tyto dva testy vyfiltrují převážnou většinu kandidátů. Proto testy rozdílů hloubek a barvy v prostoru HSV nemají na výslednou rychlost až takový vliv, jelikož pracují pouze s malou podmnožinou kandidátů.

7.4 Non-maxima suppression

Posledním krokem evaluační kaskády je aplikace non-maxima suppression na detekce, které prošly všemi výše zmíněnými testy. Verifikované templaty jsou akumulovány ze všech různých míst v obraze napříč všemi úrovněmi obrazové pyramidy. Vzhledem k tomu, že různé pohledy jednoho objektu vypadají skoro stejně a více objektů může být poměrně podobných, jsou unikátní detekce identifikovány opětovným uchováváním kandidátů majících nejvyšší skóre s_{nms} a následným odstraněním detekcí, které se s nimi překrývají.

Toto skóre lze vypočítat jako $s_{\text{nms}} = m\frac{a}{s}$, kde m je verifikační skóre definované výše (22), s je aktuální měřítko obrazové pyramidy a a je oblast objektu, kterou zabírá ve zkoumaném templatu, vztažena relativně vzhledem k jeho velikosti. Vážení výsledného skóre vzhledem k velikosti objektu upřednostňuje detekce, které toho vypovídají o scéně více (např. pokud je hrnek zobrazený ze strany s viditelnou rukojetí, bude preferován template, kde je daná rukojeť viditelná, namísto těch, u kterých je zakryta). Odfiltrované detekce jsou následně poslány do fáze precizního odhadu 3D pózy spolu s jejich výchozími pózami, které jsou přiřazené každému templatu ve fázi trénování.



Obrázek 20: Ukázka výpočtu non-maxima suppression, scéna před aplikací NMS (druhá zleva), scéna po aplikaci NMS (třetí zleva).

Princip non-maxima suppression spočívá ve výpočtu velikosti oblasti, která je překrývána dvěma různými okny detekcí. Pokud je překrytí dostatečně velké, ponechá se ve výsledném seznamu pouze detekce s vyšším skóre. K tomu je však nejprve nutné provést sestupné seřazení seznamu všech detekcí podle jejich skóre m. Dále se zvolí první detekce w_1 mající nejvyšší skóre, která je následně odstraněna ze seznamu. Ta je poté použita pro porovnání oblasti se všemi zbývajícími prvky v seznamu. V každém kroku algoritmus počítá velikost překrývané oblasti, která se poté porovnává s velikostí okna detekce w_1. Pokud je poměr překrytí o vyšší než předem definovaný práh t_o (výchozí hodnota $t_0 = 0,6$), okno dané detekce je ze seznamu odstraněno a algoritmus pokračuje dále, dokud není seznam prázdný. Výsledkem algoritmu je seznam detekcí s nejvyšším skóre, které se vzájemně nepřekrývají.

Velikost překrývané oblasti je znázorněna na obrázku 20. Lze ji vypočítat se znalostí souřadnic levého horního a pravého dolního bodu okna detekce. Dle vztahu (23), kde x_{wt_1}, y_{wt_1}, x_{wb_1}, y_{wb_1} jsou souřadnice levého horního a pravého dolního bodu okna w_1 (detekce s nejvyšším skóre), x_{wt_i}, y_{wt_i}, x_{wb_i}, y_{wb_i} jsou souřadnice levého horního a pravého dolního bodu aktuálně porovnávaného okna detekce a w_o, h_o představují výšku a šířku překrývané oblasti, která je sjednocením obou zpracovávaných oken.

$$\begin{aligned} x_1 &= \min(x_{bw_1}, x_{bw_i})\,, & x_2 &= \max(x_{tw_1}, x_{tw_i})\,, \\ y_1 &= \min(y_{bw_1}, y_{bw_i})\,, & y_2 &= \max(y_{tw_1}, y_{tw_i})\,, \\ w_o &= \max(0, x_1 - x_2)\,, & h_o &= \max(0, y_1 - y_2)\,. \end{aligned} \tag{23}$$

Pro získání hledaného poměru překrytí α_o, je použit vztah (24):

$$\alpha_o = (h_o w_o)/(h_{\mathrm{w}_1} w_{\mathrm{w}_1}), \tag{24}$$

kde h_{w_1}, w_{w_1} představují výšku a šířku okna w_1, tedy detekce s aktuálně nejvyšším skóre. Výsledný algoritmus implementovaný v jazyce C++ je přiložen níže.

```
1  void nms(std::vector<Match> &matches, float maxOverlap) {
2      // Seřazení všech kandidátů podle jejich skóre sestupně
3      std::sort(matches.rbegin(), matches.rend());
4
5      std::vector<Match> pick;
6      std::vector<int> suppress(matches.size()); // Indexy kandidátů k odebrání
7      std::vector<int> idx(matches.size());  // Indexy bounding boxů, které je
           nutné zkontrolovat
8      std::iota(idx.begin(), idx.end(), 0);  // Vyplnění pole idx hodnotami 0 -
           idx.size()
9
10     while (!idx.empty()) {
11         // Výběr prvního elementu s nejvyšším skóre
12         cv::Rect &firstMatchBB = matches[idx[0]].objBB;
13
14         // Uložení indexu do pole výsledných kandidátů, a jeho odebrání z pole
               kandidátů, které je nutné zkontrolovat
15         suppress.push_back(idx[0]);
16         pick.push_back(firstMatchBB);
17
18         // Kontrola překrytí prvního kandidáta se všemi ostatními bouding boxy
19         for (size_t i = 1; i < idx.size(); i++) {
20             cv::Rect bb = matches[idx[i]].objBB();
21             int x1 = std::min<int>(bb.br().x, firstMatchBB.br().x);
22             int x2 = std::max<int>(bb.tl().x, firstMatchBB.tl().x);
23             int y1 = std::min<int>(bb.br().y, firstMatchBB.br().y);
24             int y2 = std::max<int>(bb.tl().y, firstMatchBB.tl().y);
25
26             // Výpočet překrávající se plochy
27             int h = std::max<int>(0, y1 - y2);
28             int w = std::max<int>(0, x1 - x2);
29             float overlap = static_cast<float>(h * w) / static_cast<float>(
                   firstMatchBB.area());
```

```
30
31              // Pokud je plocha větší než práh, kandidát s menším skóre je
                    odstraněn
32              if (overlap > maxOverlap) {
33                  suppress.push_back(idx[i]);
34              }
35          }
36
37          // Odstranění všech kandidátů, kteří neprošli testem překrytí
38          idx.erase(std::remove_if(idx.begin(), idx.end(), [&suppress](int v) ->
                bool {
39                      return std::find(suppress.begin(), suppress.end(), v) !=
                            suppress.end();
40                  }), idx.end());
41          suppress.clear();
42      }
43
44      matches.swap(pick);
45  }
```

Výpis 3: Algoritmus aplikace non-maxima suppression.

8 Precizní odhad 3D pózy

Precizní odhad 3D pózy objektů v hloubkových obrazech představuje značný problém v mnoha aplikacích počítačového vidění. Typickým úkolem těchto algoritmů je určení orientace hledaného objektu, která je relativní vzhledem k nějakému souřadnému systému. Tuto informaci lze následně využít např. v robotice pro manipulaci s reálnými objekty. Póza objektu je definována kombinací rotace a pozice. Často se však tento pojem používá také pouze pro popis rotace.

V případě objektů bez textury nelze odhadnout jejich pózu s využitím state of the art technik [24] a [25]. Ty využívají SIFT [26] pro detekci a popis lokálních vlastností objektů, které jsou robustní, invariantní vůči posuvu, rotaci, velikosti a částečně vůči osvětlení scény. Pokud objekty nemají dostatečně výraznou texturu, je vhodné k určení jejich pózy využít hloubkových obrazů.

Precizní určení 3D pózy objektů v hloubkových obrazech většinou reprezentuje poslední krok detekčních algoritmů (jako je tomu i v případě této diplomové práce). Tyto postupy jsou schopné detekovat hrubý odhad výchozí pózy objektů, která je následně upřesněna. Jelikož je po detekci známa počáteční póza objektu, velmi populární metodou pro její upřesnění je ICP (více v sekci 4). Algoritmus ICP představuje zlatý standard pro geometrické zarovnání dvou množin bodů, jejichž relativní póza je přibližně známá. Avšak pokud jsou daleko od sebe, nebo mají malé překrytí, ICP může generovat velké množství nesprávných korespondencí. Tato situace se jen zhoršuje v případech, kdy je obraz postižen šumem. Výsledkem toho může být, že ICP snadno uvízne v lokálním minimu a její výkon závisí z velké míry na kvalitě určení výchozí pózy.

Obrázek 21: Vizualizace precizního odhadu 3D pózy, vstupní scéna (první zleva), výchozí pózy, které poskytuje detekční algoritmus (druhé zleva), pózy po optimalizaci (třetí zleva).

Práce autorů Rusinkiewicze a Levoye [15] představuje mnoho variant ICP, které se těmto problémům snaží vyhnout. Většina z nich však vyžaduje časově náročné ladění parametrů vzhledem ke konkrétní aplikaci. V rámci této diplomové práce se proto pro upřesnění 3D pózy objektů využívá optimalizační metody založené na Particle Swarm Optimization (PSO). PSO je optimalizační technika inspirovaná chováním hejna ptáků při hledání potravy. Snaží se optimalizovat daný problém iterativním vylepšováním populace jedinců, kteří se navzájem ovlivňují s cílem nalézt správné řešení. PSO se ukázalo jako efektivní přístup při řešení jiných problémů s určením 3D pózy [27], [28]. Zároveň vyžaduje nastavení pouze malého množství vstupních parametrů a málo kalkulací účelové funkce, dokud nezačne konvergovat ke správnému řešení.

8.1 Popis metody

Metoda, která je v této práci zpracována, dokáže odhadnout pózu objektu, jehož pozice a orientace v prostoru je přibližně známa. Na vstupu očekává model objektu, jeho výchozí pózu $\{R_0, t_0\}$ (ta je již známa z předchozí fáze evaluační kaskády), vstupní hloubkový obraz a vnitřní parametry kamery hloubkového senzoru. Výstupem je poté upřesněná póza objektu $\{R, t\}$. Obě pózy jsou v souřadném systému daného senzoru.

Modely jednotlivých objektů mohou pocházet z 3D skenů nebo CAD modelů, které odpovídají reálným objektům. Každý model M se skládá z množiny uspořádaných vertexů V, jejich odpovídajících normál U a uspořádané množiny indexů G, které definují jednotlivé trojúhelníky.

Tato metoda postupně generuje kandidáty póz $\{R_i, t_i\}$. Ty následně vyhodnocuje s pomocí vyrenderovaných syntetizovaných obrázků M, které vytvářejí hloubkové mapy D_i. Účelová funkce poté vrací skóre $o(i)$, které definuje podobnost mezi obrázky D_i a vstupním hloubkovým obrazem D s využitím hloubkových dat, hran a orientací povrchových normál. PSO je následně použito k prohledávaní definovaného prostoru pózy a minimalizaci chyby optimalizací účelové funkce s cílem nalézt pózu, která nejvíce odpovídá vstupnímu obrazu.

8.2 Inicializace

Výchozí póza, kterou poskytuje výsledek detekčního algoritmu, může být celkem hrubá. Projekce modelu v této póze definuje zarovnaný 2D bounding box b. Pro zmírnění možných nepřesností je nejprve b proporcionálně zvětšeno vzhledem k vzdálenosti objektu od kamery. Zároveň se předpokládá, že b zachycuje většinu, ne-li celou, projekci cílového objektu ve vstupním obraze.

Poté je před samotným spuštěním optimalizačního algoritmu nutné připravit několik pomocných struktur. Vstupní obraz D je (s cílem redukovat šum kamerového senzoru) filtrován mediánovým filtrem o velikosti masky 5×5. Z hloubkových obrazů jsou poté vyextrahovány povrchové normály, které tvoří obraz N stejným způsobem, jaký je již popsán v sekci 7.2.3. V obou případech jsou vadné pixely obrazu D ignorovány a nepřispívají k výslednému výpočtu účelové funkce. Následně jsou z hloubkových obrazů extrahovány hrany E aplikací Sobelova operátoru s prahováním (jak je již uvedeno v sekci 7.1). Nakonec je z E vypočítána distanční mapa T, která je uchována pro pozdější výpočty.

8.3 Renderování kandidátů póz

Renderovací proces simuluje hloubkové obrazy D_i jednotlivých kandidátů póz $\{R_i, t_i\}$, které jsou vyobrazeny vůči černému pozadí. Projekční matice v OpenGL je vytvořena ze znalosti vnitřních parametrů kamery reálného senzoru pomocí následující rovnice:

$$\begin{bmatrix} 2\frac{f_x}{w} & -2\frac{\gamma}{w} & \frac{-2x_0+w+2}{w} & 0 \\ 0 & -2\frac{f_y}{h} & \frac{-2y_0+h+2}{h} & 0 \\ 0 & 0 & q & q_n \\ 0 & 0 & -1 & 0 \end{bmatrix},$$

kde γ je zkosení obrazu, f_x, f_y ohnisková vzdálenost osy x, y; x_0, y_0 jsou souřadnice bodu ve středu kamery (v pixelech), w a h představují šířku a výšku obrazu v pixelech, q a q_n lze získat s využitím následující rovnice:

$$d = c_f - c_n\,, \quad q = \frac{-(c_f + c_n)}{d}\,, \quad q_n = -2\frac{(c_f + c_n)}{d}\,,$$

kde c_f a c_n představují zadní a přední ořezovou rovinu projekční matice v OpenGL. Každý hloubkový obraz poté odpovídá tomu, kdy by byl objekt izolován na cílové poloze v aproximované póze.

Součástí každého templatu datasetu je jeho výchozí póza, která je reprezentována rotační maticí a translačním vektorem $\{R_0, t_0\}$. Aplikace této transformace na odpovídající model jej umístí do polohy a orientace v referenčním rámečku senzoru, kterým byl daný objekt zachycen ve fázi trénování. Jednotliví kandidáti póz $\{R_i, t_i\}$ jsou parametrizováni relativně vzhledem k této výchozí póze s využitím relativního posunu a „in-place" rotace kolem centroidu c vrcholů V daného modelu. Model je tedy nejprve posunut o $-c$ do středu souřadného systému, kde je na něj aplikována rotace R_i, a následně posunut zpět o c. Rotační matice R je produktem primitivních rotací kolem 3 os $R_i = R_x(\alpha_i)R_y(\beta_i)R_z(\gamma_i)$. Každý vrchol v množiny V daného modelu tudíž musí podstoupit následující transformaci:

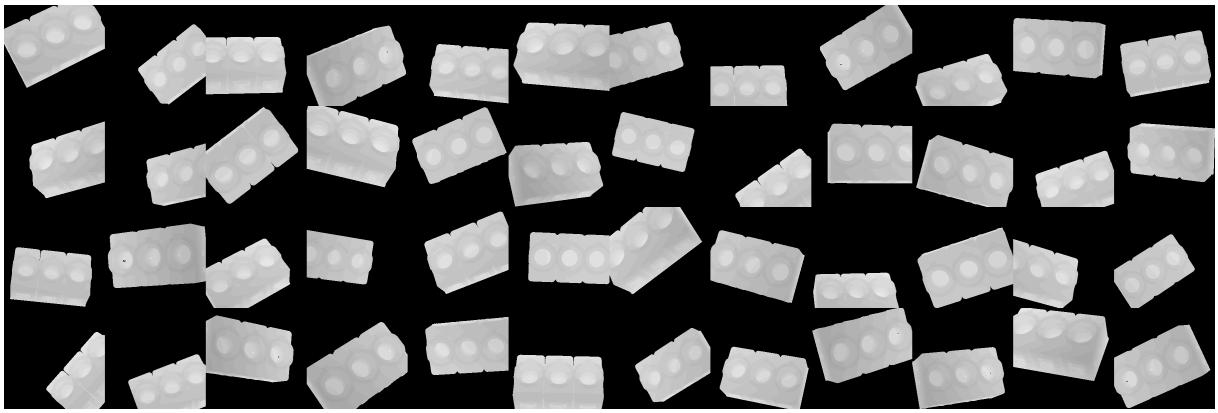
$$R_i(v\ - c) + c + t_i\,.$$

S cílem vyhnout se opakovaným výpočtům jsou transformace kandidáta pózy a výchozí pózy modelu kombinovány do jedné rovnice:

$$R_i R_0 v\ + R_i(t_0 - c) + c + t_i\,. \tag{25}$$

Jednotlivé komponenty rotační matice jsou parametrizovány Eulerovými úhly, přičemž translační vektor využívá euklidovských souřadnic. V rámci každé pózy je nejprve na daný model aplikovaná transformace s využitím rovnice (25). Mimo výše zmíněný hloubkový obraz D_i, jsou dále generovány obrazy povrchových normál N_i a hrany detekované v hloubkových obrazech E_i. S cílem redukovat množství komunikace mezi GPU a CPU jsou hloubkové mapy i povrchové normály vyrenderovány najednou do 4-kanálového obrazu. První tři kanály odpovídají normovaným souřadnicím normálových vektorů obrazu N_i, přičemž poslední obsahuje hloubková data. Při každé iteraci optimalizačního algoritmu je tak redukován počet nutných výměn dat mezi

GPU a CPU, což kladně přispívá k rychlosti algoritmu.



Obrázek 22: Vizualizace 48 výchozích kandidátů póz.

8.4 Vyhodnocení kandidátů póz

Každá póza je vyhodnocena vzhledem k tomu, jak moc se podobá vstupnímu obrazu. V ideálním případě by byl obraz modelu, vyrenderovaný na správné póze, identický. Účelová funkce $o(\cdot)$ vrací skóre $o(i)$, které definuje míru podobnosti mezi hodnotami v hloubkové mapě, povrchovými normálami a hranami v obrazech D (N, E) a D_i (N_i, E_i).

Hodnoty hloubkových dat jsou v obou obrazech přímo porovnávány v rámci jednotlivých n párů pixelů. Rozdíly hodnot těchto párů jsou akumulovány a definují ztrátovou funkci d_i:

$$d_i = \sum_{k=1}^{n} \frac{1}{|\delta_k| + 1}\,, \tag{26}$$

kde $|\delta_k|$ je rozdíl hodnot jednotlivých párů, která je nastavena na ∞ (při implementaci na maximální velikost datového typu `float`), pokud hodnota překročí definovaný práh t_k ($t_k = 20$ mm) s cílem vyhnout se porovnávání s pozadím. Pro stejné páry pixelů je provedeno porovnání povrchových normál:

$$u_i = \sum_{k=1}^{n} \frac{1}{|\gamma_k| + 1}\,, \tag{27}$$

kde γ_u představuje úhel mezi dvěma normálami, získaný jako skalární součin obou vektorů. V případě obou testů se porovnávají pouze pixely, které se nacházejí na hranici vyrenderovaného modelu. Zároveň vadné pixely vstupních hloubkových obrazů D nepřispívají k výsledku těchto funkcí. Detekované hrany obrazu E_i jsou porovnávány s využitím předpočítané distanční mapy T_i. Pokud m je celkový počet edgelů v obraze E_i, pak každé hraně j odpovídá vzdálenost ϵ_i od nejbližšího edgelu v obraze E, kterou lze vyhledat s využitím distanční mapy T na pozici j:

$$e_i = \sum_{j=1}^{m} \frac{1}{\epsilon_j + 1}\,. \tag{28}$$

Každá rovnice (26), (27) a (28) pracuje s dvěma uspořádanými množinami párů pixelů, jeden z každého obrazu D (N, E) a D_i (N_i, E_i). Jelikož d_i, u_i a e_i mají jiné rozsahy hodnot, výsledná kombinovaná hodnota účelové funkce je dána jejich násobkem:

$$o(i) = -d_i u_i e_i \,,$$

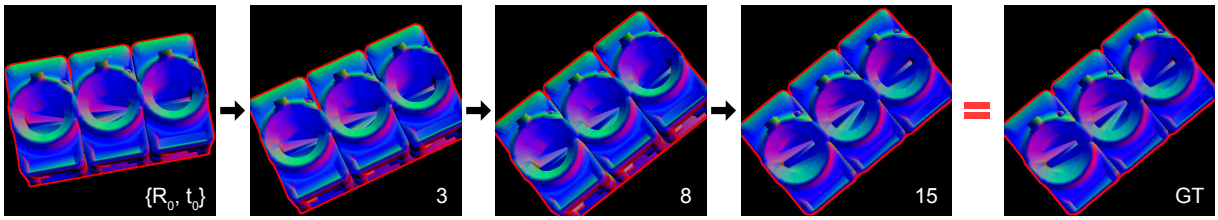
kde d_i je záporné, aby se zajistilo, že optimální hodnoty odpovídají hledání minima, jelikož d_i, u_i a e_i jsou vždy kladné. Sumy jednotlivých rozdílů $|\delta_k|$, $|\gamma_k|$ a ϵ_j upřednostňují pózy, které mají největší prostorové překrytí se vstupním obrazem. Z toho vyplývá, že hodnota účelové funkce se postupně zlepšuje s přibývajícím množstvím pixelů, které se navzájem v jednotlivých obrazech shodují. Naopak pózy se špatným překrytím nevracejí dobré skóre.

Při vyhodnocení jednotlivých kandidátů může nastat situace, kdy jsou jednotlivé pózy porovnávány s pozadím nebo překrývajícími se povrchy. Tomu se lze vyhnout porovnáváním pouze pixelů, které jsou uvnitř daného bounding boxu b. Renderované pózy, které jsou pouze částečně umístěné uvnitř b, proto vrací slabé skóre a algoritmus nemá tendenci konvergovat k nesprávným řešením.

8.5 Prohledávání prostoru pózy pomocí PSO

Prohledávací prostor pro precizní určení 3D pózy je omezen v 6D okolí kolem výchozí pózy $\{R_0, t_0\}$. Každá dimenze prohledávaného prostoru má definované hranice, které vytvářejí hyperkrychli, jejíž střed je zarovnaný na střed výchozí pózy. PSO je použito pro prohledávání tohoto prostoru s cílem optimalizovat hodnotu účelové funkce a najít řešení, které nejvíce odpovídá vstupnímu obrazu.

PSO stochasticky vyvíjí populaci částic (ty jsou reprezentovány jednotlivými kandidáty póz $\{R_i, t_i\}$), které iterativně prohledávají definovaný prostor v několika generacích [28]. Každá částice postupně upravuje své parametry, které jsou reprezentovány komponentami translačního vektoru (t_x, t_y, t_z) a Eulerovými úhly rotace kolem každé osy ($R_x(\alpha)$, $R_y(\beta)$, $R_z(\gamma)$). Dále je pro každý parametr definován vektor rychlosti v_i, který udává, jak rychle se budou jednotlivé částice pohybovat napříč prohledávaným prostorem.



Obrázek 23: Ukázka průběhu PSO při optimalizaci pózy objektu v populaci 50 jedinců (v závorce u obrázku je uvedena aktuální generace).

Algoritmus pracuje ve třech základních krocích, které se opakují, dokud není splněna některá z ukončovacích podmínek (maximální počet iterací, minimální chyba). Nejprve je vypočítána

hodnota účelové funkce (skóre $o(i)$), která značí fitness každé částice v populaci. Ta si poté zapamatuje svůj nejlepší výsledek a zároveň je v celé populaci nalezen globální nejlepší výsledek. Poté je aktualizována pozice každé částice s využitím vektoru rychlosti. Nakonec dojde k výpočtu nových hodnot vektoru rychlosti pomocí rovnice:

$$v_i(t+1) = wv_i(t) + c_1 r_1[x_{\text{pbest}}(t) - x_i(t)] + c_2 r_2[x_{\text{gbest}}(t) - x_i(t)]\,,$$

kde w, c_1, c_2 jsou konfidenční koeficienty ($0 \leq w \leq 1{,}2$; $0 \leq c_1, c_2 \leq 2$), r_1, r_2 náhodná čísla ($0 \leq r_1, r_2 \leq 1$), $x_{\text{pbest}}(t)$ je nejlepší nalezený kandidát pro každou částici p_i v generaci t a $x_{\text{gbest}}(t)$ představuje nejlepší globální řešení v čase t. Pozice částice je následně aktualizována s využitím rovnice:

$$x_i(t+1) = x_i(t) + v_i(t+1)\,. \tag{29}$$

Pro zajištění co nejlepšího pokrytí prohledávaného prostoru je prvotní populace vygenerována s pomocí 6D Sobol sekvence, která zajišťuje jejich dobré prostorové rozmístění. PSO takto postupně aktualizuje pozice všech částic na konci každé generace s využitím rovnice (29). Zde je nutné zajistit, aby se jednotlivé částice nedostaly mimo prohledávaný prostor, a pokud tato situace nastane, vrátit je na původní pozici. Částice se navzájem ovlivňují, dokud není dosaženo ukončovacího kritéria (maximální počet generací), na jehož konci algoritmus vrátí nejlepšího kadidáta pózy $\{R_i, t_i\}$, který se nachází pod částicí x_{gbest}.

8.5.1 Parametry PSO při implementaci

Prvotní populace póz je generována s pomocí 6D Sobol sekvence, kde $t_{ix}, t_{iy} \in \langle -25; 25\rangle$ a $t_{iz} \in \langle -75; 125\rangle$ z důvodů větší tolerance vůči faktoru změny velikosti obrazové pyramidy. Komponenty rotační matice jsou definovány v rozsahu $\alpha, \beta, \gamma \in \langle -0{,}5; 0{,}5\rangle$ s cílem pokrýt větší množství možných rotací výsledné pózy. Vektory rychlosti jsou generovány náhodně vynásobením čísla $f \in \langle 0; 1\rangle$ faktory $f_{tx}, f_{ty} = 10$; $f_{tz} = 25$; $f_\alpha, f_\beta, f_\gamma = 0{,}15$.

Při každé generaci je před aktualizací pozice částice kontrolováno, zda se nenachází mimo vymezený prostor. Ten je pro rotaci definován v intervalu $\langle -0{,}6; 0{,}6\rangle$, pro posun v osách x, y v intervalu $\langle -40; 40\rangle$ a pro posun v ose z v intervalu $\langle -150; 250\rangle$.

Parametry PSO jsou dále následující: $c_1, c_2 = 0{,}15$. Tyto hodnoty se po důkladném testování ukázaly jako ideální, jelikož umožňují částicím důkladně prozkoumat prohledávaný prostor v několika prvních generacích a přitom zajistit včasnou migraci k nejlepšímu výsledku. Konfidenční koeficient w je dále rozdělen na dva koeficienty $w_1 = 0{,}9; w_2 = 0{,}97$, které udávají různé faktory snížení vektoru rychlosti zvlášť pro translaci a rotaci. Zde je snaha rychleji snížit rychlost posunu, jelikož správná pozice hledaného objektu je nalezena velmi rychle, a naopak zachovat větší rychlost změn v rotaci s cílem prozkoumat větší množství více odlišných póz kandidátů. Počet generací a velikost populace je rovna 50.

9 Závěrečná měření

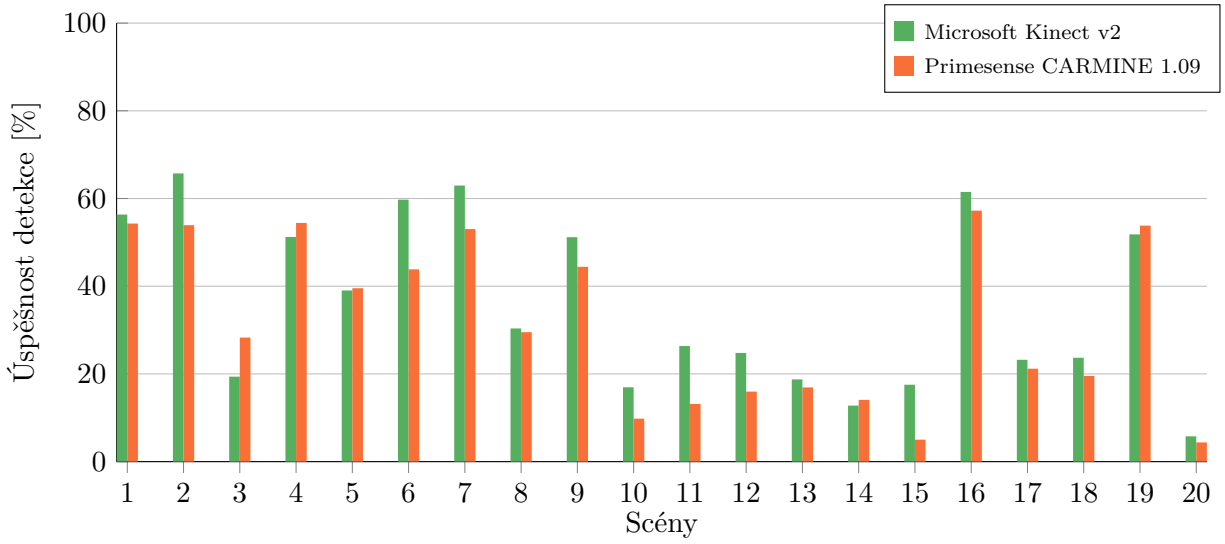
Výše popsaná metoda byla důkladně otestována s využitím datasetu popsaného v sekci 4. Ten obsahuje 20 testovacích scén, které se liší ve složitosti od nejjednodušších s částečnou okluzí po složité scény s velkým množstvím objektů. Každá scéna je reprezentována 504 zarovnanými RGB a RGB-D obrazy spolu s anotacemi v podobě ground truth 6D póz a bounding boxů, spolu s id objektů, které se v nich vyskytují.

Výsledná měření jsou rozdělena do dvou částí. V první části je ověřena úspěšnost detekce objektů s využitím dat obou senzorů (Microsoft Kinect v2 a Primesense CARMINE 1.09) ve dvou datových sadách. První sada na vstupu vždy očekává danou scénu a trénovací data pouze těch objektů, které se v ní vyskytují. Druhá sada již pracuje s celou databází trénovacích objektů (30) pro každou scénu s cílem ověřit vliv objektů, které se ve scéně nenacházejí, na přesnost a rychlost detekčního algoritmu. Druhá část se zabývá vyhodnocením precizního určení 3D pózy objektů na výstupech detekčního algoritmu.

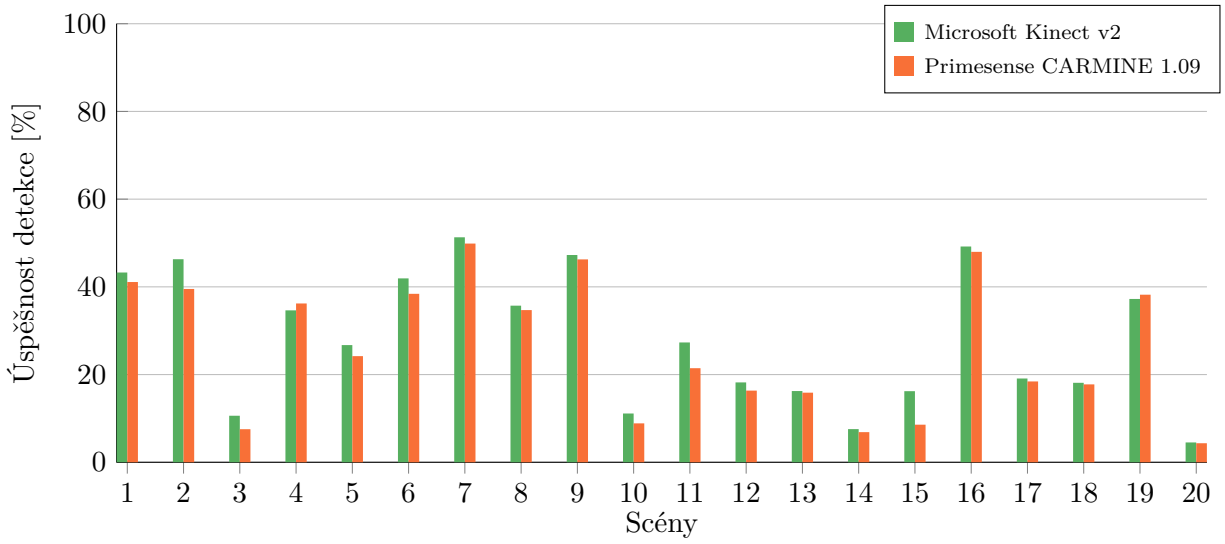
Úspěšnost detekce byla ověřena s využitím výše zmíněných ground truth bounding boxů a Jaccard indexu, který definuje míru překrytí detekovaných oken s grouth truth daty a umožňuje zjistit počet TP, FN, FP ve scéně. Úspěšnost je poté vypočítána jako F1 skóre:

$$p = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \quad r = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \quad \mathrm{F1} = 2\frac{pr}{p + r}.$$

Algoritmus byl spuštěn pro každou scénu pětkrát s cílem eliminovat dopady vnějších vlivů. Výsledné hodnoty, které lze vidět na níže uvedených grafech, jsou tedy aritmetickým průměrem z těchto pěti měření. Všechna měření byla provedena na notebooku vybaveném procesorem Intel Core i7 s frekvencí 3.1 GHz (Turbo Boost až 4.1 GHz), 16GB RAM a grafickou kartou Radeon Pro 560.



Obrázek 24: Úspěšnost algoritmu obou senzorů na první datové sadě pro jednotlivé scény.



Obrázek 25: Úspěšnost algoritmu obou senzorů na druhé datové sadě pro jednotlivé scény.

Na obrázku 24 lze pozorovat, že úspěšnost se v rámci jednotlivých scén velice liší. Lépe se v testu umístilo prvních 9 scén, kde jsou objekty vyobrazeny vůči černému pozadí. Zbylé scény $10 - 20$ si vedou mnohem hůře. Objekty v těchto scénách jsou umístěny na dřevěném povrchu s výraznou texturou spolu s jinými objekty, které nejsou v databázi, což stěžuje podmínky detekčnímu algoritmu. Z výsledků je zřejmé, že si detekční algoritmus s těmito obtížnými scénami dokáže jen těžko poradit.

Naopak velkým překvapením jsou scény 16 a 19, které i přes ztížené podmínky dosahují dobrých výsledků. To je dáno především tím, že jsou ve scéně vyhledávány převážně kulaté objekty, jejichž body příznaků na hranách vytváří kruhy. Poté, i pokud jsou některé objekty detekovány na nesprávných místech, neprojdou testem orientací gradientů ve fázi ověření kandidátů.

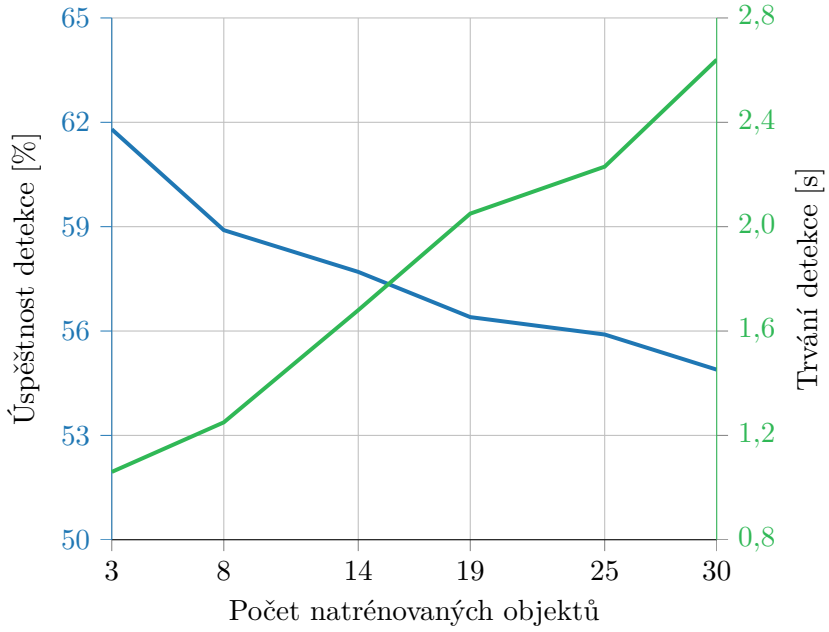
Dále je nutné zmínit, že ve scénách, kde se vyskytují objekty s rovnými plochami, vznikalo mnoho FP detekcí na místech, kde se vyskytují ArUco značky (ty slouží pro kalibraci kamery). Ze samotné podstaty testů lze proti tomuto jen těžko bojovat. Detekovaná plocha je rovná a neměnná, což umožní její průchod všemi testy povrchových normál (orientace povrchu je shodná s podstavou detekované scény). Zároveň orientace gradientů ArUco značek odpovídají hranám určitých templatů těchto objektů. Nakonec jejich bílá barva umožní splnění předpokladů pro průchod posledního testu, jelikož všechny objekty v databázi jsou bílé.

Obrázek 25 poté ukazuje úspěšnost detekce, kdy je natrénovaná pouze jedna množina obsahující všechny objekty v databázi. Lze pozorovat, že v průměru došlo ke snížení úspěšnosti o 8%, na některých scénách až o 20%. Nakonec obrázek 26 znázorňuje sub-lineární časovou složitost v závislosti na počtu natrénovaných objektů. Z obrázku je viditelné, že při desetinásobku počtu objektů se časová složitost navýšila pouze třikrát.

Tyto výsledky nelze přímo porovnat s úspěšností původní práce [3], kde autor dosahuje až 95,4% úspěšnosti. Zde byl však použit syntetický dataset obsahující velmi odlišené objekty

různých barev. Odpadá zde tudíž nutnost řešit šum a nepřesnost reálných hloubkových senzorů. Zároveň různorodost barev jednotlivých objektů kladně napomáhá k redukci špatných detekcí v posledním testu ověření kandidátů.

V práci [16] byla stejná metoda aplikována na použitý dataset. Autor zde uvádí 67,2% úspěšnost při stejné metodice, která je aplikována v první datové sadě (na vstupu jsou pouze objekty, které se v dané scéně nacházejí). Této úspěšnosti se již některé scény vlastní implementace algoritmu blíží. Ve výše uvedeném článku se však autor nezmiňuje o hodnotách mnoha koeficientů použitých při detekci, které mohou značně ovlivnit úspěšnost detekce. Zároveň neuvádí, zda jsou pro každou scénu definované jiné hodnoty koeficientů nebo zůstávají stejné po celou dobu měření.



Obrázek 26: Závislost úspěšnosti a časové náročnosti na počtu objektů v databázi.

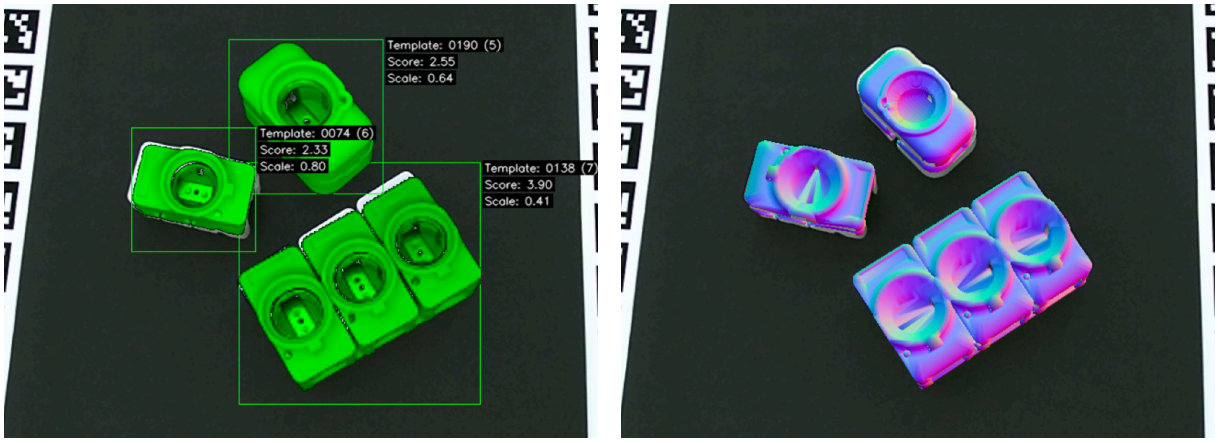
9.1 Vyhodnocení precizního určení 3D pózy

Přestože dataset obsahuje ground truth pózy pro každou scénu, tak po důkladném zkoumání a mnoha konzultacích s vedoucím práce se na základě nedostatku informací o použitém datasetu nepodařilo zjistit transformaci, která by posunula model objektu do detekovaného bounding boxu v kameře snímané scény. Póza je proto upřesněna s využitím vnitřních parametrů senzoru, kterým byl zachycen objekt při trénování. Na funkčnost algoritmu tento problém nemá vliv. Pro jeho vyhodnocení však není možné použít výše zmíněných ground truth póz a výsledek je možné posoudit pouze vizuálně.

Na obrázku níže jsou znázorněny výsledky precizního určení 3D pózy pro scénu č. 2. PSO se ukázalo v syntetických testech jako velice efektivní i v případech, kdy byla výchozí detekce

relativně nepřesná. I v těchto situacích bylo schopné nalézt správnou pózu detekovaného objektu již v několika prvních generacích. V případě reálných dat byly výsledky spíše smíšené.

Jelikož jsou vstupní obrazy datasetu postiženy šumem, nelze v nich v mnoha případech zcela jasně detekovat edgely hloubkových obrazů, které se používají při výpočtu účelové funkce. To vede k tomu, že PSO není vždy schopné určit přesnou pózu objektu. Zároveň v případech, kdy nelze ve scéně spolehlivě detekovat povrchové normály (větší množství neplatných pixelů), PSO má tendenci konvergovat ke špatným řešením, která vrací lepší skóre.



Obrázek 27: Výsledek precizního určení 3D pózy pomocí PSO, detekované pózy (vlevo), pózy po upřesnění (vpravo).

9.2 Časová náročnost jednotlivých operací

V následující tabulkách jsou znázorněny časové náročnosti výpočtů jednotlivých výše popsaných částí detekční kaskády na třech scénách, které se liší ve své složitosti. Každá hodnota je průměrem měření napříč všemi obrázky dané scény vydělena počtem scén (tedy průměr na jednu scénu). Při implementaci algoritmu bylo použito OpenMP pro paralelizaci. Algoritmus je tudíž schopen efektivně využít více jader procesoru ve všech kritických krocích aplikace, což značně snižuje jeho časovou náročnost.

Časy jednotlivých kroků jsou v následujících tabulkách rozděleny do dvou částí: detekce a precizní určení 3D pózy objektu. První část využívá při výpočtech pouze CPU, přičemž druhá do výpočtu zapojuje i GPU.

Tabulka 3: Časové náročnosti jednotlivých částí algoritmu pro Microsoft Kinect v2.

Akce	**Časová náročnost [ms]**		
	Scéna 2	**Scéna 7**	**Scéna 19**
Počet objektů	3	8	8
Načtení scény a vytvoření obrazových pyramid	291	298	309
Detekce objektivity	23	32	34
Výběr kandidátů (hashování)	156	1034	1587
Ověření kandidátů	82	236	357
Non-maxima suppression	$<< 1$	5	1
Detekce celkem	**552**	**1605**	**2287**
Generování prvotní populace	$<< 1$	$<< 1$	$<< 1$
Vyrenderování kandidátů póz (50 póz v 50 generacích)	10123	15264	14466
Výpočet účelové funkce	3233	3756	3125
Precizní určení 3D pózy celkem	**13356**	**19020**	**17591**
Celkem	**13908**	**20625**	**19878**

Z výše uvedené tabulky lze pozorovat, že nejvíce času v části detekce zabírá fáze výběru kandidátů. Přestože se v této fázi počítá pouze 5 hodnot pro každou tabulku v předem definovaných tripletech ($5 \cdot 100$), pracuje s největším množstvím templatů, což dělá tuto fázi časově nejnáročnější. Časová složitost fáze ověření kandidátů a non-maxima suppression se následně odvíjí od počtu kandidátů, kteří projdou předchozí fází. Těch je více s přibývající složitostí scény.

Precizní určení 3D pózy se skládá ze tří hlavních kroků. V první fázi je na CPU vygenerována prvotní populace kandidátů póz. Poté je v každé generaci pro každou pózu vyrenderován normálový a hloubkový obraz na GPU pro něž je následně vypočítána účelová funkce na CPU (jejíž výpočet je paralelizován a časová náročnost záleží na velikosti detekovaného bounding boxu). V této části je nejnáročnější právě vyrenderování jednotlivých objektů. Přestože vyrenderování jednoho objektu trvá méně než 1 ms (záleží na velikosti bounding boxu), při PSO je celkový počet renderů roven NGP, kde N je počet detekovaných objektů, P velikost populace a G počet generací. To dělá tuto fázi precizního určení 3D pózy časově nejnáročnější.

Tabulka 4: Časové náročnosti jednotlivých částí algoritmu pro Primesense CARMINE 1.09.

Akce	**Časová náročnost [ms]**		
	Scéna 2	**Scéna 7**	**Scéna 19**
Počet objektů	3	8	8
Načtení scény a vytvoření obrazových pyramid	301	294	293
Detekce objektivity	31	36	34
Výběr kandidátů (hashování)	469	1088	1402
Ověření kandidátů	245	351	474
Non-maxima suppression	$<< 1$	5	20
Detekce celkem	**846**	**1774**	**2223**
Generování prvotní populace	$<< 1$	$<< 1$	$<< 1$
Vyrenderování kandidátů póz (50 póz v 50 generacích)	11232	16124	15496
Výpočet účelové funkce	3397	4176	3645
Precizní určení 3D pózy celkem	**14629**	**20300**	**19141**
Celkem	**15475**	**22074**	**21364**

Ve srovnání s Microsoft Kinect v2 je Primesense CARMINE 1.09 pomalejší. To je způsobeno tím, že hloubková data tohoto senzoru jsou vyhlazenější. Neobsahují tudíž tolik šumu, což přispívá k tomu, že fází výběru kandidátů projde více templatů. Zároveň je úspěšnost detekce tohoto senzoru méně úspěšná. Ve výsledku je proto více FP detekcí, které musí precizní určení 3D pózy zpracovat.

10 Závěr

V této diplomové práci byla nastudována problematika detekce a sledování objektů. Jedná se především o objekty bez textury, které se vyskytují zejména v průmyslu a robotických aplikacích. První část diplomové práce je zaměřena na teoretický popis algoritmů a technik, které jsou zde následně použity. Dále je uveden popis použitého datasetu včetně úprav, které na něm byly provedeny.

Druhá část se zaměřuje na popis a implementaci detekčního algoritmu s precizním odhadem 3D pózy objektů, která vychází z nedávné práce [**4**]. Implementovaná metoda je založena na principu posuvného okénka v kombinaci s obrazovými pyramidami s kaskádovým zpracováním detekovaných oken. Na scénu je nejprve aplikována detekce objektivity, která vrátí pouze okna, ve kterých se s největší pravděpodobností nachází hledaný objekt. Sub-lineární časová složitost vzhledem k počtu natrénovaných objektů je zajištěna ve fázi výběru kandidátů s využitím předem natrénovaných hashovacích tabulek a aplikace procesu hlasování. Pro každé okno je poté vybrána malá množina kandidátů s největším počtem hlasů. Ti jsou následně ověřeni pomocí pěti po sobě jdoucích testů, lišících se ve složitosti, které ověřují kandidáty v různých modalitách (orientace gradientů, hloubková data, povrchové normály a barva v prostoru HSV). Aplikace non-maxima suppression v poslední fázi zajistí ponechání pouze těch kandidátů póz objektů, které mají nejlepší shodu.

Výchozí póza těchto objektů, kterou poskytuje předchozí fáze detekční kaskády, je dále upřesněna s využitím stochastických metod v podobě PSO. V této fázi algoritmus postupně renderuje odpovídající 3D modely detekovaných objektů s pomocí OpenGL v několika náhodně rozmístěných pózách kolem výchozí pózy. Vhodnost každé pózy je ohodnocena s využitím definované účelové funkce. PSO je poté použito k optimalizaci výsledku účelové funkce a nalezení nejlepšího řešení v podobě pózy, která nejvíce odpovídá vstupním datům.

Nakonec bylo provedeno závěrečné zhodnocení na výše zmíněném datasetu. Rychlost algoritmu se vzhledem k množství provedených operací ukázala jako dostačující, a to zejména díky využití paralelizace v podobě OpenMP a mnoha optimalizačních vylepšení uvedených v samotné práci. Úspěšnost detekce byla otestována na dvou testovacích sadách s využitím trénovacích dat dvou dostupných senzorů (Microsoft Kinect v2 a Primesense CARMINE 1.09). Složitost použitého datasetu se projevila na úspěšnosti výsledné detekce, kde se nepodařilo dosáhnout stejné procentuální úspěšnosti jako v práci [16]. Zde však autor neuvádí hodnoty jednotlivých koeficientů, a proto nelze tyto výsledky přímo srovnávat. U precizního určení 3D pózy měly na výslednou přesnost vliv především hloubkové obrazy, u nichž nebylo mnohdy možné spolehlivě detekovat hrany, což mělo negativní vliv na výpočet účelové funkce a konvergenci ke správnému řešení.

Při implementaci algoritmu se průběžně vyskytovaly nové problémy, které nebyly v původním článku dostatečně popsány, či se o nich autor vůbec nezmiňoval, a bylo nutné je řešit. To značně zpomalilo celkový průběh práce na implementaci tohoto algoritmu. Z jeho úspěšnosti

vyplývá, že je zde stále prostor pro zlepšení. Jedno z možných vylepšení spočívá v efektivnějším generování pozic tripletů, které značně ovlivňují výsledek algoritmu. Dalším může být (vzhledem k použitému datasetu) nahrazení testu barvy v prostoru HSV jiným testem, který by dokázal objekty od sebe a od pozadí lépe rozlišit.

Literatura

[1] TANG, Jie; MILLER, Stephen; SINGH, Arjun; ABBEEL, Pieter. A textured object recognition pipeline for color and depth image data. In: *2012 IEEE International Conference on Robotics and Automation*. 2012, s. 3467–3474. ISSN 1050-4729. Dostupné z DOI: 10.1109/ICRA.2012.6224891.

[2] TUYTELAARS, Tinne; MIKOLAJCZYK, Krystian. *Local Invariant Feature Detectors:A Survey*. Local Invariant Feature Detectors:A Survey. Now Foundations a Trends, 2008. ISBN 9781601981387. Dostupné z DOI: 10.1561/0600000017.

[3] HODAN, Tomas; ZABULIS, Xenophon; LOURAKIS, Manolis I. A.; OBDRZALEK, Stepan; MATAS, Jiri. Detection and fine 3D pose estimation of texture-less objects in RGB-D images. In: *IROS*. IEEE, 2015, s. 4421–4428. ISBN 978-1-4799-9994-1. Dostupné také z: http://dblp.uni-trier.de/db/conf/iros/iros2015.html#HodanZLOM15.

[4] SWAROOP, Paridhi; SHARMA, Neelam. An Overview of Various Template Matching Methodologies in Image Processing. 2016, roč. 153, s. 8–14.

[5] MAHALAKSHMI, T; MUTHAIAH, R; SWAMINATHAN, P. An overview of template matching technique in image processing. 2012, roč. 4, s. 5469–5473.

[6] COX, G. S. *Template Matching and Measures of Match in Image Processing*. 1995. Dostupné také z: http://www.dip.ee.uct.ac.za/imageproc/pattern/cox7_95.ps.gz.

[7] ALEXE, B.; DESELAERS, T.; FERRARI, V. Measuring the Objectness of Image Windows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2012, roč. 34, č. 11, s. 2189–2202. ISSN 0162-8828. Dostupné z DOI: 10.1109/TPAMI.2012.28.

[8] MARR, David; HILDRETH, E. Theory of Edge Detection. *Proceedings of the Royal Society of London Series B*. 1980, roč. 207, s. 187–217.

[9] LINDEBERG, Tony. Edge Detection and Ridge Detection with Automatic Scale Selection. *International Journal of Computer Vision*. 1998, roč. 30, č. 2, s. 117–156. ISSN 1573-1405. Dostupné z DOI: 10.1023/A:1008097225773.

[10] SUJATHA, P.; SUDHA, K. K. Performance Analysis of Different Edge Detection Techniques for Image Segmentation. *Indian Journal of Science and Technology*. 2015, roč. 8, č. 14. ISSN 0974 -5645. Dostupné také z: http://www.indjst.org/index.php/indjst/article/view/72946.

[11] KIMMEL, R.; BRUCKSTEIN, A.M. Regularized Laplacian Zero Crossings as Optimal Edge Integrators. *International Journal of Computer Vision*. 2003, roč. 53, č. 3, s. 225–243. ISSN 1573-1405. Dostupné z DOI: 10.1023/A:1023030907417.

[12] CANNY, J. A Computational Approach to Edge Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* 1986, roč. 8, č. 6, s. 679–698. ISSN 0162-8828. Dostupné z DOI: 10.1109/TPAMI.1986.4767851.

[13] CHEN, Yang; MEDIONI, Gerard. Object Modelling by Registration of Multiple Range Images. *Image Vision Comput.* 1992, roč. 10, č. 3, s. 145–155. ISSN 0262-8856. Dostupné z DOI: 10.1016/0262-8856(92)90066-C.

[14] BESL, P. J.; MCKAY, N. D. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1992, roč. 14, č. 2, s. 239–256. ISSN 0162-8828. Dostupné z DOI: 10.1109/34.121791.

[15] RUSINKIEWICZ, S.; LEVOY, M. Efficient variants of the ICP algorithm. In: *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*. 2001, s. 145–152. Dostupné z DOI: 10.1109/IM.2001.924423.

[16] HODAN, T.; HALUZA, P.; OBDRZALEK, S.; MATAS, J.; LOURAKIS, M.; ZABULIS, X. T-LESS: An RGB-D Dataset for 6D Pose Estimation of Texture-Less Objects. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2017, s. 880–888. Dostupné z DOI: 10.1109/WACV.2017.103.

[17] HINTERSTOISSER, Stefan; LEPETIT, Vincent; ILIC, Slobodan; HOLZER, Stefan; BRADSKI, Gary; KONOLIGE, Kurt; NAVAB, Nassir. Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes. In: LEE, Kyoung Mu; MATSUSHITA, Yasuyuki; REHG, James M.; HU, Zhanyi (ed.). *Computer Vision – ACCV 2012*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, s. 548–562. ISBN 978-3-642-37331-2.

[18] HINTERSTOISSER, S.; CAGNIART, C.; ILIC, S.; STURM, P.; NAVAB, N.; FUA, P.; LEPETIT, V. Gradient Response Maps for Real-Time Detection of Textureless Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2012, roč. 34, č. 5, s. 876–888. ISSN 0162-8828. Dostupné z DOI: 10.1109/TPAMI.2011.206.

[19] DROST, B.; ULRICH, M.; NAVAB, N.; ILIC, S. Model globally, match locally: Efficient and robust 3D object recognition. In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, s. 998–1005. ISSN 1063-6919. Dostupné z DOI: 10.1109/CVPR.2010.5540108.

[20] CHOI, C.; CHRISTENSEN, H. I. 3D pose estimation of daily objects using an RGB-D camera. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, s. 3342–3349. ISSN 2153-0858. Dostupné z DOI: 10.1109/IROS.2012.6386067.

[21] CAI, Hongping; WERNER, Tomas; MATAS, Jiri. Fast Detection of Multiple Textureless 3-D Objects. In: CHEN, Mei; LEIBE, Bastian; NEUMANN, Bernd (ed.). *Computer Vision Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, s. 103–112. ISBN 978-3-642-39402-7.

[22] CHENG, M. M.; ZHANG, Z.; LIN, W. Y.; TORR, P. BING: Binarized Normed Gradients for Objectness Estimation at 300fps. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, s. 3286–3293. ISSN 1063-6919. Dostupné z DOI: 10.1109/CVPR.2014.414.

[23] FACCIOLO, Gabriele; LIMARE, Nicolas; MEINHARDT-LLOPIS, Enric. Integral Images for Block Matching. 2014, roč. 4, s. 344–369.

[24] ROMEA, Alvaro Collet; TORRES, Manuel Martinez; SRINIVASA, Siddhartha. The MOPED framework: Object recognition and pose estimation for manipulation. *International Journal of Robotics Research*. 2011, roč. 30, č. 10, s. 1284 –1306.

[25] LOURAKIS, Manolis; ZABULIS, Xenophon. Model-Based Pose Estimation for Rigid Objects. In: CHEN, Mei; LEIBE, Bastian; NEUMANN, Bernd (ed.). *Computer Vision Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, s. 83–92. ISBN 978-3-642-39402-7.

[26] LOWE, David G. Object Recognition from Local Scale-Invariant Features. In: *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*. Washington, DC, USA: IEEE Computer Society, 1999, s. 1150–. ICCV '99. ISBN 0-7695-0164-8. Dostupné také z: http://dl.acm.org/citation.cfm?id=850924.851523.

[27] IASON OIKONOMIDIS, Nikolaos Kyriazis; ARGYROS, Antonis. Efficient model-based 3D tracking of hand articulations using Kinect. In: *Proceedings of the British Machine Vision Conference*. BMVA Press, 2011, s. 101.1–101.11. ISBN 1-901725-43-X. Dostupné z DOI: 10.5244/C.25.101.

[28] ROSA, Stefano; TOSCANA, Giorgio; BONA, Basilio. Q-PSO: Fast Quaternion-Based Pose Estimation from RGB-D Images. *Journal of Intelligent & Robotic Systems*. 2017. ISSN 1573-0409. Dostupné z DOI: 10.1007/s10846-017-0714-3.

A Přílohy na CD

Adresářová struktura na přiloženém CD je následující:

- `src` – zdrojové kódy implementace výše popsaného algoritmu,
- `doc` – dokumentace k výše uvedeným zdrojovým kódům vygenerovaná pomocí nástroje Doxygen.